دانشگاه آزاد اسلامی

واحد شبستر

مروری بر برخی از روش های بهینه سازی هوشمند

تدوین و گردآوری

- حبیب مطیع قادر
- شهریار لطفی
- میر مهدی سید اسفهلان

سرشناسه	: حبیب مطیع قادر، ۱۳۶۲
عنوان و نام پدیدآور	: مروری بر برخی از روش‌های بهینه‌سازی هوشمند/ تدوین و گردآوری حبیب مطیع قادر، شهریار لطفی و میر مهدی سید اسفهلان.
ویراستار ادبی	: شهریار لطفی
مشخصات نشر	: شبستر- دانشگاه آزاد اسلامی (شبستر)، ۱۳۸۹
مشخصات ظاهری	۲۱۵ص: مصور، جدول، نمودار.
شابک	: ۹۷۸-۹۶۴-۱۰-۰۳۷۱-۷
وضعیت فهرست نویسی	: فیپا
یادداشت	کتابنامه
موضوع	: مهندسی تولید، الگوهای ریاضی
شناسه افزوده	: لطفی، شهریار، ۱۳۵۴
شناسه افزوده	: سید اسفهلان ، میر مهدی ، ۱۳۶۲
شناسه افزوده	دانشگاه آزاد اسلامی، واحد شبستر
رده بندی کنگره	: ۱۳۸۹ ۴م۶م/TS۱۷۶
رده بندی دیویی	:۶۵۸/۵۰۱۵۱۱۸
شماره کتابشناسی ملی	:۲۱۸۳۹۸۲

دانشگاه آزاد اسلامی – واحد شبستر

نام کتاب: مروری بر برخی از روش‌های بهینه‌سازی هوشمند

تدوین و گردآوری: حبیب مطیع قادر، شهریار لطفی و میر مهدی سید اسفهلان.

چاپ اول: شبستر، ۱۳۸۹

ناشر: دانشگاه آزاد اسلامی – واحد شبستر

شمارگان: ۲۰۰۰ جلد

تعداد صفحات و قطع: ۲۱۵، وزیری

حروفچینی: دانشگاه آزاد اسلامی (شبستر)

طرح روی جلد: حسن همایی سرشت

شابک: ۹۷۸-۹۶۴-۱۰-۰۳۷۱-۷

چاپ: انتشارات دانشگاه آزاد اسلامی

قیمت: ۶۰۰۰۰ ریال

حبیب مطیع قادر

تقدیم به پدر و مادر عزیز
و خواهر مهربانم

شهریار لطفی

تقدیم به همسر و پسر عزیزم

میر مهدی سید اسفهلان

تقدیم به پدر و مادر عزیز
و همسر مهربانم

تقدیر و تشکر

بر خود لازم می‌دانیم که از جناب آقایان دکتر سعید پارسا، دکتر جمشید باقرزاده و دکتر سید مصطفی کلامی که در گردآوری این کتاب ما را همراهی نمودند، کمال تقدیر و تشکر را داشته باشیم.

پیشگفتار

وجود مسایل پیچیده علمی منجر می‌شود تا سراغ روش‌های بهینه‌سازی رفته و مساله مورد نظر را به وسیله آن‌ها حل کرد. با توجه به زمان‌بر بودن و پیچیدگی روش‌های دقیق از روش‌های بهینه‌سازی هوشمند استفاده می‌شود. تاکنون روش‌های بهینه‌سازی متعددی معرفی شده‌اند که از مهم‌ترن آن‌ها می‌توان به الگوریتم‌های تکاملی، الگوریتم تپه نوردی، الگوریتم شبیه ساز سرد کردن فلزات، الگوریتم بهینه‌سازی انبوه ذرات، الگوریتم جستجوی ممنوع، الگوریتم بهینه‌سازی مورچه‌ها، خودکارهای یادگیر و غیره اشاره نمود.

هدف از گردآوری این کتاب معرفی برخی از مهم‌ترین روش‌های بهینه‌سازی است. مسایل بهینه‌سازی در علوم مختلفی اعم از فنی و مهندسی، علوم پایه و انسانی کاربرد فراوانی دارند. به علت پر اهیمت بودن این موضوع تلاش نموده‌ایم تا در این راستا کتابی را گردآوری نماییم. در این کتاب شش روش مختلف بهینه‌سازی معرفی خواهیم کرد. این شش روش عبارتند از: الگوریتم ژنتیک، الگوریتم شبیه‌ساز سرد کردن فلزات، الگوریتم بهینه‌سازی مورچه‌ها، الگوریتم بهینه‌سازی انبوه ذرات، یادگیری تقویتی و خودکارهای یادگیر. این روش‌ها از جمله روش‌های پر طرفدار و پر کاربرد است که در بیشتر مسایل بهینه‌سازی مورد استفاده قرار گرفته‌اند. مخاطبین اصلی این کتاب کسانی هستند که در یکی از رشته‌های تحصیلی علوم کامپیوتر، ریاضیات کاربردی، الکترونیک، الکتروتکنیک، کنترل، صنایع، مکاترونیک و علوم انسانی فعالیت دارند. البته این کتاب می‌تواند در رشته‌های مرتبط دیگری که به نحوی با مسایل بهینه‌سازی روبرو هستند نیز مورد استفاده قرار گیرد. از مخاطبین گرامی مشتاقانه درخواست می‌شود تا در صورت مشاهده هرگونه ایراد و کاستی در کتاب آن را از طریق آدرس پست الکترونیکی habib_moti@yahoo.com به نویسندگان اعلام نمایند.

فهرست مطالب

فصل اول: مقدمه‌ای بر بهینه‌سازی و روش‌های آن

فصل دوم: الگوریتم‌های تکاملی

فصل سوم: الگوریتم شبیه‌ساز سرد کردن فلزات

فصل چهارم: الگوریتم بهینه‌سازی مورچه‌ها

فصل پنجم: الگوریتم بهینه‌سازی انبوه ذرات

فصل ششم: یادگیری تقویتی

فصل هفتم: خودکارهای یادگیر

پیوست الف: استفاده از الگوریتم ژنتیک برای حل مساله زمان‌بندی چند پردازنده‌ای

پیوست ب: استفاده از خودکار یادگیر برای حل مساله زمان‌بندی چند پردازنده‌ای

فصل اول

مقدمه‌ای بر بهینه‌سازی و روش‌های آن

۱-۱ مقدمه

بهینه‌سازی فرآیندی است که برای بهتر کردن چیزی دنبال می‌شود. فکر، ایده و یا طرحی که به وسیله یک دانشمند یا یک مهندس مطرح می‌شود، طی روال بهینه‌سازی بهتر می‌شود [۱]. در هنگام بهینه‌سازی، شرایط اولیه با روش‌های مختلف مورد بررسی قرار می‌گیرد و اطلاعات به دست آمده، برای بهبود بخشیدن به یک فکر یا روش مورد استفاده قرار می‌گیرند. بهینه‌سازی ابزاری ریاضی است که برای یافتن پاسخ بسیاری از پرسش‌ها در خصوص چگونگی راه حل مسایل مختلف به کار می‌رود [۵، ۶].

در بهینه‌سازی از یافتن بهترین جواب برای یک مساله صحبت به میان می‌آید. لفظ *بهترین* به طور ضمنی بیان می‌کند که بیش از یک جواب برای مساله مورد نظر وجود دارد که البته دارای ارزش یکسانی نیستند. تعریف بهترین جواب، به مساله مورد بررسی، روش حل و هم‌چنین میزان خطای مجاز وابسته است. بنابراین نحوه فرمول‌بندی مساله نیز بر چگونگی تعریف بهترین جواب تاثیر مستقیم دارد. برخی از مسایل جواب‌های مشخصی دارند؛ بهترین بازیکن یک رشته‌ی ورزشی، طولانی‌ترین روز سال و پاسخ یک معادله‌ی دیفرانسیل معمولی درجه اول از مثال‌هایی هستند که می‌توان از آن‌ها به عنوان مسایل ساده نام برد. در مقابل، برخی از مسایل دارای جواب‌های بیشینه[۱] یا کمینه[۲]ی متعددی هستند

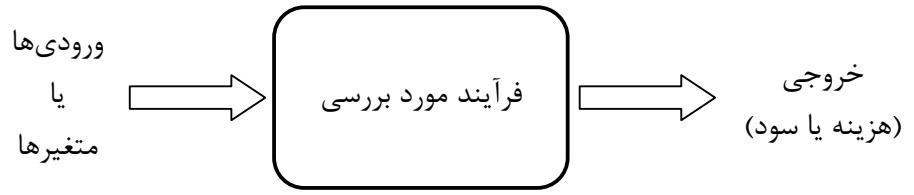
---

[۱] maximum

[۲] minimum

که به نام نقاط بهینه یا اکسترمم[۱] شناخته می‌شوند، و به احتمال بهترین جواب یک مفهوم نسبی خواهد بود. بهترین اثر هنری، زیباترین منظره و گوش‌نوازترین قطعه‌ی موسیقی از مثال‌هایی هستند که می‌توان برای این گونه مسایل بیان کرد [۵-۱۰].

بهینه‌سازی، تغییر دادن ورودی‌ها و خصوصیات یک دستگاه، فرآیند ریاضی و یا آزمایش تجربی است به نحوی که بهترین خروجی یا نتیجه به دست بیاید (شکل ۱-۱). ورودی‌ها متغیرهای فرآیند یا تابع مورد بررسی هستند که به نام‌های *تابع هدف*[۲]، *تابع هزینه*[۳] و یا *تابع برازندگی*[۴] نامیده می‌شود. خروجی نیز به صورت هزینه، سود و یا برازندگی تعریف می‌شود [۵-۱۰]. در این نوشتار نیز، مطابق با بسیاری از نوشتارهای مرتبط با موضوع، تمام مسایل بهینه‌سازی به صورت کمینه‌سازی مقدار یک تابع هزینه در نظر گرفته شده‌اند. به راحتی می‌توان نشان داد که هر نوع مساله بهینه‌سازی را می‌توان در قالب یک مساله کمینه‌سازی تعریف نمود.



شکل ۱-۱: فرآیند یا تابعی که بهینه‌سازی می‌شود. در بهینه‌سازی ورودی‌ها یا متغیرها به نحوی تغییر داده می‌شوند که خروجی مطلوب به دست آید [۱].

گاهی اوقات مساله بهینه‌سازی به نام *برنامه‌ریزی ریاضی*[۵] نیز خوانده می‌شود. یک مساله بهینه‌سازی از نظر ریاضی به صورت زیر بیان می‌شود [۶-۱۰]:

$$\begin{aligned} &\text{minimize} \quad f(x) \\ &\text{subject to} \quad g_i(x) \le b_i, \quad i = 1, 2, \ldots, m \end{aligned} \qquad (۱-۱)$$

که در آن $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$، متغیر اصلی و مستقل مساله است که با تغییر دادن آن، مقدار کمینه برای تابع هدف پیدا می‌شود. تابع هدف به صورت $f : \mathbb{R}^n \to \mathbb{R}$ تعریف شده است و دارای مقدار حقیقی می‌باشد. مجموعه‌ی توابع $g_i : \mathbb{R}^n \to \mathbb{R}$ نیز تعریف شده‌اند تا قیودی به صورت نامساوی به وسیله آن‌ها بیان شود. اعداد حقیقی سمت راست این نامساوی‌ها، یعنی b_iها، حدود

---

[۱] extremum
[۲] objective function
[۳] cost function
[۴] fitness function
[۵] mathematical programming

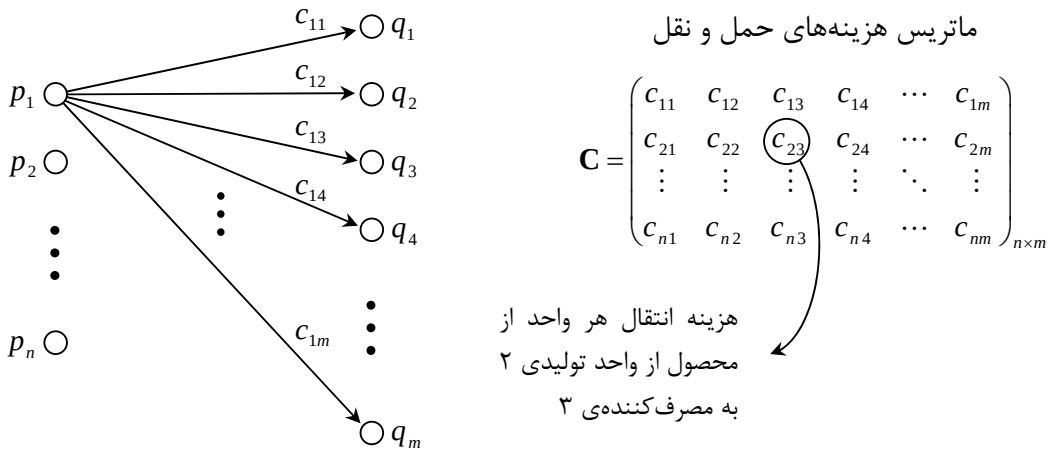
نامساوی‌ها هستند. می‌توان زیرمجموعه‌ای از $\mathbb{R}^n$ به نام Ω یافت، به نحوی که قیود نامساوی برای همه‌ی اعضای این مجموعه برقرار باشند. در اصطلاح Ω *مجموعه‌ی شدنی*[۱] نامیده می‌شود [۶-۸، ۹].

مساله‌ای که به صورت (۱-۱) تعریف شده است یک مساله کلی است و راه حلی برای حالت کلی آن وجود ندارد. هر گاه تابع هدف $f(.)$ و قیود مساله همگی خطی باشند، یک مساله برنامه‌ریزی خطی مطرح می‌شود. تابع $h(.)$ خطی است اگر و فقط اگر رابطه‌ی زیر به ازای هر α و β که اسکالر (حقیقی یا مختلط) هستند، برقرار باشد [۶-۱۰]:

$$h(\alpha x + \beta y) = \alpha h(x) + \beta h(y) \qquad (۲-۱)$$

روش‌های متعددی برای حل مسایل بهینه‌سازی خطی وجود دارند، که موارد استفاده‌های فراوانی در علوم مختلف دارند [۶-۸، ۱۰]. یکی از مسایل مشهور در زمینه‌ی بهینه‌سازی خطی، *مساله حمل و نقل*[۲] است [۶-۵].

مساله حمل و نقل: محصولی در n واحد تولیدی به مقدار p_1، p_2، ... و p_n تولید می‌شود. هم‌چنین m مصرف‌کننده با میزان مصرف q_1، q_2، ... و q_m وجود دارند، که می‌بایست همه‌ی نیاز آن‌ها به وسیله تولیدکننده‌ها تامین شود. فرض بر این است که هزینه انتقال هر واحد از کالای مورد نظر از واحد تولیدی iاُم به مصرف‌کننده‌ی jاُم، معلوم و برابر با c_{ij} می‌باشد. برای داشتن کم‌ترین هزینه حمل و نقل، هر کدام از واحدهای تولیدی چه قدر از نیاز هر مصرف‌کننده را تأمین کنند؟ (شکل ۱-۲)



شکل ۱-۲: مساله حمل و نقل [۱].

---

[۱] feasible set

[۲] transportation problem

۱-۲ انواع مسایل بهینه‌سازی

می‌توان مسایل بهینه‌سازی را از دیدگاه‌های مختلف به دسته‌های متعددی تقسیم‌بندی کرد. نمونه‌ای از این تقسیم‌بندی‌ها در ادامه توضیح داده شده‌اند.

الف) بهینه‌سازی با سعی و خطا[۱] و بهینه‌سازی روی تابع: سعی و خطا فرآیندی است که در آن متغیرهای ورودی تغییر داده می‌شوند و اطلاعات دقیق در خصوص نحوه تاثیر هر متغیر بر خروجی در دست نیست. به عنوان مثال نحوه تنظیم آنتن گیرنده‌ی تلویزیونی یک بهینه‌سازی با سعی و خطا است. حتی یک مهندس آنتن، تنها توانایی حدس زدن نحوه عملکرد آنتن در جهات مختلف را ندارد. بسیاری از کشفیات مهم بشر، نتیجه‌ی بهینه‌سازی‌های توام با سعی و خطا بوده است. به عنوان مثال می‌توان به کشف و تصفیه‌ی پنی‌سیلین به عنوان یک آنتی‌بیوتیک اشاره نمود. در مقابل نوع دیگری از بهینه‌سازی وجود دارد که در آن ماهیت مساله، به صورت فرمول دقیق در دست است و می‌توان با روش‌های ریاضی به سراغ این‌گونه مسایل رفت [۵].

ب) بهینه‌سازی تک‌بُعدی و بهینه‌سازی چندبُعدی: اگر فقط یک متغیر در مساله بهینه‌سازی وجود داشته باشد، بهینه‌سازی تک‌بعدی نامیده می‌شود. در مقابل مسایلی که دارای بیش از یک متغیر باشند، مسایل بهینه‌سازی چندبعدی خوانده می‌شوند [۵-۷، ۹].

پ) بهینه‌سازی پویا[۲] و بهینه‌سازی ایستا[۳]: اگر تابع یا فرآیندی که مورد بهینه‌سازی واقع می‌شود، تابعی از زمان باشد و با گذشت زمان تغییر یابد بهینه‌سازی را پویا می‌نامند. در مقابل بهینه‌سازی روی مسایلی که گذشت زمان تغییری روی آن‌ها به وجود نمی‌آورد، بهینه‌سازی ایستا خوانده می‌شود. به عنوان مثال، پیدا کردن بهترین مسیر برای رانندگی در شهر برای رسیدن از یک هدف به یک مقصد معین، می‌تواند یک مساله بهینه‌سازی ایستا باشد. می‌توان با استفاده از یک نقشه، بهترین مسیر ممکن را به دست آورد. اما این مساله در عمل به این سادگی نیست. در عمل می‌بایست عوامل دیگری نیز در نظر گرفته شوند. در ساعات مختلف شبانه‌روز میزان[۴] ازدحام متغیر است و لذا بهترین مسیر ممکن است در زمان‌های مختلف تغییر پیدا کند [۵-۷، ۱۰].

ت) بهینه‌سازی گسسته و بهینه‌سازی پیوسته: اگر ماهیت متغیرهای مساله بهینه‌سازی پیوسته باشد، آن را پیوسته می‌نامند. در مقابل اگر مقادیری که متغیرهای مساله اختیار می‌کنند، مقادیر محدود و شمارا باشند، مساله را گسسته می‌نامند. یک نوع بسیار مهم از مسایل گسسته، *مسایل جایگشت*[۵]

---

[۱] trial and error optimization

[۲] dynamic optimization

[۳] static optimization

[۴] traffic

[۵] permutation problems

هستند. هدف از حل این نوع از مسایل، انجام یک انتخاب از بین یک مجموعه از گزینه‌های قابل انتخاب است که ترتیب انتخاب نیز مهم است [۵-۶].

ث) بهینه‌سازی مقید و بهینه‌سازی بدون قید: در برخی مسایل متغیرها نمی‌توانند هر مقداری را اختیار کنند و می‌بایست مقادیر متغیرها یک مجموعه از شرایط را برآورده کنند. این شرایط را قید و مسایل توام با قید را مقید می‌نامند. هر مساله که قیدی در آن وجود نداشته باشد، بدون قید خوانده می‌شود [۵-۱۰].

ج) بهینه‌سازی کمینه‌جو[۱] و بهینه‌سازی تصادفی[۲]: برخی از روش‌ها از یک نقطه‌ی مشخص در فضای جستجو شروع می‌کنند و با استفاده از قوانینی که پایه در ریاضیات و هندسه دارند، جواب‌های بهتری به دست می‌آورند. این نوع الگوریتم‌ها سرعت بسیار بالایی در هم‌گرایی دارند، اما به راحتی در کمینه‌ها یا بیشینه‌های محلی گرفتار می‌شوند. در این روش‌ها نحوه ایجاد جواب‌های بعدی از روی جواب‌های فعلی، روندی مشخص و معلوم دارد. در مقابل روش‌های تصادفی، از الگوهای احتمالی برای ایجاد جواب‌های بهتر استفاده می‌کنند و به این ترتیب پیش‌بینی عملکرد الگوریتم ساده نیست. سرعت هم‌گرایی این نوع از الگوریتم‌ها در مقایسه با الگوریتم‌های کمینه‌جو، کم‌تر است اما احتمال گرفتار شدن در نقاط بهینه‌ی محلی نیز کم‌تر می‌شود و امید بیشتری برای یافته شدن نقطه‌ی بهینه‌ی سراسری وجود دارد [۲، ۵].

۱-۳ روش‌های بهینه‌سازی کمینه‌جو

در زیر تعدادی از روش‌های بهینه‌سازی کمینه‌جو معرفی می‌شود.

۱-۳-۱ بهینه‌سازی تحلیلی

در ریاضیات روش‌هایی برای حل مسایل بهینه‌سازی وجود دارند. این روش‌ها برای برخی از توابع هزینه، معتبر و به سادگی قابل استفاده هستند. بسیاری از مسایل پیچیده نیز با اندکی تغییر و تحول، قابل تبدیل به توابعی هستند که به وسیله روش‌های تحلیلی می‌توان مقدار کمینه یا بیشینه‌ی آن‌ها را پیدا کرد [۵-۸].

نقاط اکسترمم هر تابعی، صفرهای مشتق مرتبه‌ی اول آن تابع هستند. به این ترتیب با مشتق‌گیری از تابع و صفر قرار دادن مقدار آن و حل معادله‌ی حاصل، می‌توان محل نقاط اکسترمم را به دست آورد. اگر علامت مشتق دوم تابع در یک نقطه‌ی اکسترمم منفی باشد، نقطه‌ی اکسترمم به دست آمده، یک نقطه‌ی کمینه‌ی محلی برای تابع مورد بررسی می‌باشد. به همین ترتیب علامت مثبت برای مشتق دوم،

---

[۱] minimum-seeking optimization

[۲] random optimization

حاکی از وجود نقطه‌ی بیشینه‌ی محلی می‌باشد. البته نکات مذکور برای توابعی با متغیر اسکالر معتبر هستند. برای توابع چند متغیره، می‌توان از معادل‌هایی که در حساب دیفرانسیل برداری تعریف شده‌اند، استفاده نمود. به عنوان مثال به جای مشتق از گرادیان تابع استفاده می‌شود [۵-۶، ۸].

روش‌های فراوانی برای بهتر کردن فرآیند بهینه‌سازی تحلیلی پیشنهاد شده‌اند. اما در کل برای مسایلی که تابع هدف آن‌ها پیچیده است و یا دارای تعریف ریاضی روشنی نیست، روش تحلیلی کارآیی چندانی نخواهد داشت. به همین دلیل روش‌هایی مبتنی بر روش تحلیلی، اما به شیوه‌ای متفاوت تعریف شده‌اند. نمونه‌ای از این روش‌ها در ادامه به صورت اجمالی معرفی خواهند شد.

۱-۳-۲ جستجوی خطی[۱]

در این روش، با شروع از یک نقطه در فضای جستجو و حرکت در یک راستا به میزان معین، نقاط بعدی به عنوان جواب‌های بهتر پیشنهاد می‌شوند. این روش، اساس کار بسیاری از روش‌های بهینه‌سازی چندمتغیره نیز می‌باشد. اگر x_k جواب یافته شده در مرحله kاُم باشد، جواب مرحله‌ی بعدی به این صورت به دست می‌آید [۶-۱۰]:

$$x_{k+1} = x_k + \alpha_k d_k \qquad (۳-۱)$$

در رابطه‌ی (۱-۳) ، α_k یک اسکالر و d_k برداری هم بُعد با x_k است. d_k جهت حرکت و میزان حرکت را مشخص می‌کنند. پیدا کردن α_k و d_k مناسب در هر مرحله، اصلی‌ترین کاری است که باید انجام شود. α_k را می‌توان از روی رابطه زیر محاسبه کرد:

$$f(x_k + \alpha_k d_k) = \min_{\alpha} f(x_k + \alpha d_k) \qquad (۴-۱)$$

که در آن $f(.)$ تابع هزینه‌ای است که هدف کمینه کردن آن است. شرط فوق فقط برای یک *جستجوی خطی دقیق*[۲] برآورده می‌شود و α_k اندازه‌ی گام بهینه خوانده می‌شود. اگر α_k به نحوی انتخاب شود که، مقدار تابع هزینه در حد قابل قبولی کم شود؛ آن‌گاه *جستجوی خطی نادقیق*[۳] و یا *قابل قبول*[۴] انجام گرفته است. در این روش فقط کافی است که اختلاف هزینه $f(x_{k+1}) - f(x_k)$ منفی و کم‌تر از یک حد معین باشد. در عمل به ندرت از روش جستجوی خطی دقیق استفاده می‌شود. زیرا یافتن مقدار بهینه برای اندازه گام، کاری به نسبت سخت و توأم با هزینه محاسباتی بسیار زیادی است.

---

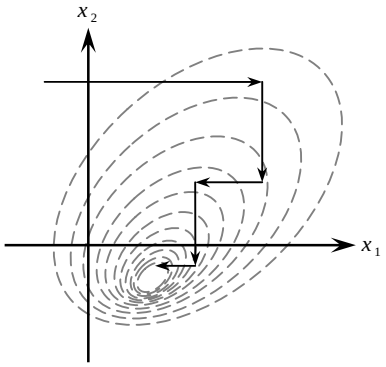
[۱] line search

[۲] exact line search

[۳] inexact line search

[۴] acceptable line search

برای تعیین جهت حرکت، d_k، روش‌های مختلفی وجود دارند. در یکی از این روش‌ها، جهت حرکت همواره در جهت یک مجموعه از بردارهای پایه و متعامد فضای جستجو در نظر گرفته می‌شود. به این ترتیب که حرکت در یک جهت تا وقتی معتبر است که منجر به کاهش هزینه شود. در غیر این صورت، جهت حرکت به یکی از جهت‌های دیگر که عمود بر جهت قبلی است، تغییر داده می‌شود. نحوه عملکرد این روش برای یک تابع در فضای دو بعدی در شکل ۱-۳ قابل مشاهده است [۵].



شکل ۱-۳ : نمودار یک تابع درجه دو در فضای دو بعدی و مسیری که در بهینه‌سازی خطی برای یافته شدن کمینه‌ی این تابع طی شده است. حرکت در هر مسیر، تا زمانی ادامه دارد که منجر به کاهش مقدار تابع شود. در غیر این صورت، مسیر حرکت عوض خواهد شد [۱].

۱-۳-۳ روش‌های نیوتونی

دسته‌ای از روش‌ها برای بهینه‌سازی بدون قید وجود دارند که اساس کار همه‌ی آن‌ها یکسان است. یکی از ساده‌ترین روش‌ها در گروه روش‌های نیوتونی [۶-۱۰]، *روش بیشترین کاهش*[۱] یا *روش گرادیان*[۲] است. فرض کنید تابع $f(.)$ حول x_k به طور یکنواخت مشتق‌پذیر باشد و $g_k \triangleq \nabla f(x_k) \neq 0$ باشد. با استفاده از دو جمله اول بسط تیلور برای $f(.)$،

$$f(x) \simeq f(x_k) + (x - x_k)^T g_k \qquad (۵-۱)$$

طبق الگوی جستجوی خطی، می‌توان نوشت $x - x_k = \alpha_k d_k$، و جهت d_k که در رابطه $d_k^T g_k < 0$ صدق کند، می‌تواند به عنوان یک جهت کاهشی استفاده شود، زیرا $f(x) < f(x_k)$ خواهد بود. با ثابت در نظر گرفتن α_k، منفی‌ترین مقدار برای $d_k^T g_k$ می‌تواند سریع‌ترین کاهش را به همراه داشته باشد. با استفاده از نامساوی کوشی-شوارتز،

$$f(x) \simeq f(x_k) + (x - x_k)^T g_k \qquad (۶-۱)$$

---

[۱] steepest descent method

[۲] gradient method

می‌توان به این نتیجه رسید که سریع‌ترین کاهش زمانی اتفاق خواهد افتاد که $d_k = -g_k$ باشد. بنابراین $-g_k$ جهت سریع‌ترین کاهش است. طرح تکراری روش گرادیان برای تولید جواب‌های بهتر به این صورت است:

$$x_{k+1} = x_k - \alpha_k g_k \qquad (۷-۱)$$

شیوه‌ی تعیین α_k نیز مشابه با روش جستجوی خطی می‌باشد و می‌تواند به صورت دقیق و یا نادقیق تعیین شود. روش گرادیان به نوع و ساختار تابع هزینه مورد بررسی بسیار حساس است و فرض مشتق‌پذیر بودن تابع هزینه برای این روش، یک فرض قوی است و استفاده از آن را به رده‌ی خاصی از توابع محدود کرده است. به همین دلیل نسخه‌های تغییر یافته‌ای از این روش ایجاد شده‌اند که قصد بر طرف کردن معایب این روش را داشته‌اند. یکی از نسخه‌های تغییر یافته‌ی روش گرادیان، روش نیوتون است. در این روش از مشتق مرتبه‌ی دوم تابع هزینه نیز برای تعیین جهت حرکت برای کاهش مقدار تابع استفاده شده است. در این روش از تقریب درجه‌ی دوم تابع برای بهینه‌سازی مقدار آن بهره گرفته شده است. با استفاده از بسط درجه‌ی دوم تابع $f(.)$ حول x_k،

$$f(x_{k+1}) = f(x_k + s) \simeq f(x_k) + \left[\nabla f(x_k)\right]^T s + \frac{1}{2} s^T \nabla^2 f(x_k) s \qquad (۸-۱)$$

برای آن که مقدار $f(x_{k+1})$ کمینه شود، می‌بایست رابطه‌ی زیر برقرار باشد:

$$x_{k+1} = x_k - \left[\nabla^2 f(x_k)\right]^{-1} \nabla f(x_k) \qquad (۹-۱)$$

که با تعریف $g_k = \nabla f(x_k)$ و $G_k = \nabla^2 f(x_k)$ می‌توان نوشت:

$$x_{k+1} = x_k - G_k^{-1} g_k \qquad (۱۰-۱)$$

جهت $-G_k^{-1}g_k$، یک جهت کاهشی است و به اصطلاح جهت کاهشی نیوتون خوانده می‌شود. مشاهده می‌شود که به جای α_k که یک اسکالر بود، در فرمول مربوط به روش نیوتون از ماتریس G_k^{-1} استفاده شده است، که به طور مستقیم از ساختار تابع مورد بررسی به دست آمده است. نتایج عملی نشان می‌دهند که به دلیل درجه‌های آزادی بیشتری که به وجود آمده است و هم‌چنین به علت وابستگی G_k^{-1} به شکل تابع، روش نیوتون کارآیی بیشتری نسبت به روش گرادیان دارد. برای روش نیوتون نیز نسخه‌های تغییر یافته‌ای وجود دارند که در هر کدام از این روش‌ها بخشی از روش نیوتون تغییر داده شده است. به عنوان مثال، در یکی از این روش‌ها برای محاسبه‌ی مشتق‌های تابع، از روش‌های عددی استفاده شده است. اما ضعف اصلی همه‌ی روش‌های نیوتونی، در استفاده از مشتق می‌باشد. لذا برای توابعی که به طور پیوسته مشتق‌پذیر نیستند و یا در اصل تابعی برای مشتق گرفتن وجود ندارد، نمی‌توان از این روش‌ها استفاده نمود.

انواع دیگری از روش‌های بهینه‌سازی عددی وجود دارند که بر اساس روش نیوتونی پایه‌ریزی شده‌اند. *روش‌های گرادیان مزدوج*[۱] و *روش‌های شبه نیوتونی*[۲] از جمله‌ی این روش‌ها هستند. در روش‌های گرادیان مزدوج، از مفهوم تعامد تعمیم یافته استفاده شده است. به این ترتیب که جهت‌های حرکت در مراحل مختلف الگوریتم، نسبت به هم‌دیگر و روی یک هسته‌ی ماتریسی معین متعامد هستند. روش‌های شبه نیوتونی نیز بیشتر در جهت رفع مشکلات مربوط به سختی محاسبات مشتق دوم برآمده‌اند و مستلزم هزینه محاسباتی کم‌تری می‌باشند [۶-۱۰].

۱-۳-۴ روش کاهشی نِلدِر- مید با اشکال غیر مرکب[۳]

روشی است که در سال ۱۹۶۵ به وسیله نِلدِر و مید معرفی شد [۵-۱۰] و در این روش نیازی به محاسبه‌ی مشتق توابع وجود ندارد. بلکه بر اساس یک سری قواعد و عملیات هندسی، جواب‌های مناسب و بهتر از روی جواب‌های پیشین تولید می‌شوند. *سیمپلِکس*[۴] یا *شکل غیر مرکب*، عبارت است از ساده‌ترین شکل هندسی که می‌توان در هر فضا ترسیم کرد و نتوان آن را به شکل بسته‌ی دیگری تجزیه نمود. سیمپلکس در فضای nبُعدی، شکلی است که $n+1$ راس و $n+1$ ضلع دارد.

در هر مرحله از روش کاهشی سیمپلکس، $n+1$ نقطه وجود دارند که یک سیمپلکس را تشکیل می‌دهند. بهترین نقطه‌ای که در سیمپلکس وجود دارد، به عنوان جواب مساله در همان مرحله است. در اولین مرحله از الگوریتم، فقط یک نقطه از سیمپلکس به نام P_0، مشخص می‌شود. باقی نقاط سیمپلکس با استفاده از رابطه زیر، ایجاد می‌شوند:

$$P_k = P_0 + c_s e_k \qquad k = 1, 2, \ldots, n \qquad (۱۱-۱)$$

در رابطه مذکور، c_s یک ضریب مقیاسی است و e_k ها نیز بردارهای واحدی در فضای جستجو هستند. هدف این الگوریتم، احاطه کردن نقطه‌ی بهینه در میان یک سیمپلکس است. مراحلی که برای بهتر کردن یک سیمپلکس دو بعدی انجام می‌شود، در شکل ۱-۴ آمده است.

عملیات جبری که در مراحل مختلف الگوریتم بیان شده‌اند، با این فرض نوشته شده‌اند که همه‌ی نقاط مورد نظر، به صورت بردارهایی در فضای $\mathbb{R}^2$ بیان شده باشند. نمودار هندسی مراحل مختلف الگوریتم در شکل ۱-۵ قابل مشاهده هستند. به همین ترتیب می‌توان قواعدی برای سیمپلکس‌های با بعد بیشتر نیز نوشت. اما تعداد انعکاس‌ها با توجه به افزایش ابعاد بیشتر می‌شوند.

---

[۱] conjugate gradient methods

[۲] quasi-newton methods

[۳] nelder-mead downhill simplex method

[۴] simplex

۱. *سیمپلکس اولیه:* سه نقطه‌ی A، B و C را که تشکیل دهنده‌ی سیمپلکس (در فضای دو بعدی، مثلث) مورد نظر هستند، در نظر بگیرید. در نظر داشته باشید که، A بهترین نقطه‌ی این مثلث است.

۲. *انعکاس:* نقطه‌ی جدیدی به نام D که حاصل انعکاس A نسبت به نقطه‌ی میانی پاره خط BC است، پیدا می‌شود. می‌توان D را از رابطه‌ی زیر به دست آورد:

$$D = B + C - A.$$

۳. *انبساط:* اگر D نقطه‌ی بهتری باشد، نتیجه می‌گیریم که حرکت در جهت درستی انجام شده است و یک گام دیگر در جهت قبلی حرکت می‌کنیم، تا نقطه‌ی E به دست آید. E از رابطه‌ی زیر قابل محاسبه است:

$$E = \frac{3(B + C) - 4A}{2}.$$

اگر E بهتر از A باشد، سیمپلکس EBC، یک سیمپلکس بهتر است. در غیر این صورت DBC را به عنوان سیمپلکس بهتر شده انتخاب می‌کنیم.

۴. *انقباض:* اگر D دارای هزینه برابر با نقطه‌ی A و یا بیشتر باشد، آن گاه دو نقطه به صورت زیر محاسبه می‌شوند:

$$F = \frac{3(B + C) - 2A}{4} \text{ و } G = \frac{2A + B + C}{4}.$$

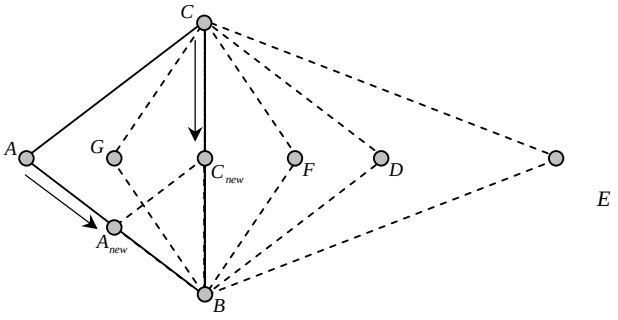
اگر بهترین نقطه از بین F یا G، بهتر از A باشد، به جای آن می‌نشیند و سیمپلکس جدیدی تشکیل می‌شود.

۵. *کوچک کردن:* اگر هیچ کدام از نقاط F یا G بهتر از A نبودند، می‌بایست سیمپلکس ABC به طرف رأس B کوچک‌تر شود به نحوی که:

$$A \leftarrow \frac{A + B}{2} \text{ و } C \leftarrow \frac{C + B}{2}.$$

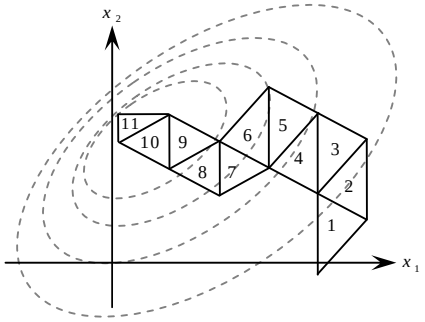
توجه کنید که در این مرحله فرض شده است که نقطه B در مقایسه با C بهتر است.

شکل ۱-۴ : الگوریتم بهینه کردن یک سیمپلکس در فضای دو بعدی [۱]. به شکل ۱-۵ توجه کنید.



شکل ۱-۵ : یک سیمپلکس دو بعدی (مثلث) که مراحل مختلف الگوریتم شکل ۱-۴ بر روی آن انجام شده است تا سیمپلکس بهتری ساخته شود [۱].

طی مراحل مختلف الگوریتم، سیمپلکس‌های بهتر و کوچک‌تری ساخته می‌شوند و اجرای الگوریتم تا جایی ادامه پیدا می‌کند که بزرگ‌ترین قطر سیمپلکس کم‌تر از یک حد معین شود. این الگوریتم، چندان سریع نیست، اما در برخورد با مسایل مختلف، عملکرد خوبی دارد. از طرفی امکان دارد این الگوریتم در نقطه‌ی بهینه‌ی محلی گرفتار شود. لذا ترکیب کردن آن با الگوریتم‌های بهینه‌سازی تصادفی، می‌تواند بسیار مفید و کارآمد باشد. به این ترتیب که مراحل میانی یا نهایی از اجرای یک الگوریتم تصادفی، به این الگوریتم سیمپلکس سپرده می‌شود. در شکل ۱-۶ نمونه‌ای از اجرای الگوریتم سیمپلکس بر روی یک تابع دو بعدی دیده می‌شود [۵].



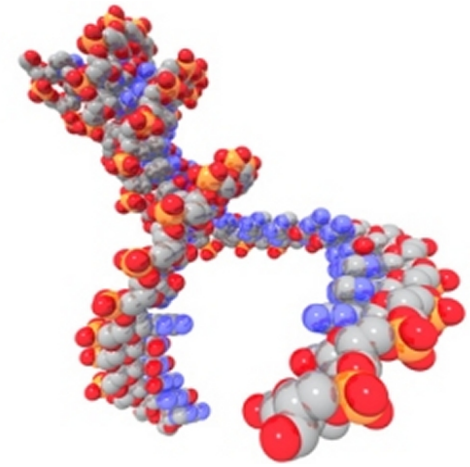
شکل ۱-۶ : نمونه‌ای از نحوه عملکرد الگوریتم کاهشی سیمپلکس بر روی یک تابع دو متغیره [۱].

۱-۴ بخش بندی فصل‌های آتی کتاب

در این فصل مفهوم بهینه‌سازی و انواع آن شرح داده شد. تعدادی از روش‌های بهینه‌سازی هوشمند که امروزه کاربرد بیشتری دارند عبارتند از:
(الف) الگوریتم ژنتیک، **(پ)** الگوریتم تپه نوردی، **(ت)** الگوریتم شبیه‌ساز حرارتی، **(ث)** الگوریتم بهینه‌سازی انبوه ذرات، **(ج)** الگوریتم بهینه‌سازی مورچه‌ها، **(د)** الگوریتم جستجوی ممنوع، **(ه)** یادگیری تقویتی، **(و)** خودکارهای یادگیر و غیره.

در این کتاب برخی از این روش‌ها به همراه مثال و کاربرد آن‌ها در مسایل مختلف شرح داده شده است. ساختار فصل‌های آتی کتاب به شرح زیر می‌باشد:

در فصل دوم الگوریتم ژنتیک به همراه چند مثال ساده معرفی شده است. درفصل سوم الگوریتم شبیه‌ساز سرد کردن فلزات به همراه یک مثال مطرح شده است. فصل چهارم الگوریتم بهینه‌سازی مورچه‌ها را به همراه مثال شرح داده است. در فصل پنجم الگوریتم بهینه‌سازی انبوه ذرات توضیح داده شده است. در فصل ششم یادگیری تقویتی به همراه مثال ارایه شده است و در فصل هفتم خودکارهای یادگیر به همراه کاربرد آن در مساله رنگ آمیزی گراف‌ها آورده شده است.



فصل دوم

الگوریتم‌های تکاملی[1]

۲-۱ مقدمه

هنگامی که لغت تنازع بقا به کار می‌رود، اغلب یک نگرش منفی به ذهن می‌آید. شاید هم‌زمان قانون جنگل به ذهن برسد. البته برای آن که بتوان آسوده خاطر شد، می‌توان این گونه پنداشت که همیشه قوی‌ترین موجودات برنده نبوده‌اند. به عنوان مثال دایناسورها با وجود جثه عظیم و قوی‌تر در طی روندی به طورکامل طبیعی ادامه نسل[2] را واگذار کردند. در حالی که موجوداتی بسیار ضعیف‌تر از آن‌ها حیات خویش را ادامه دادند. به ظاهر طبیعت بهترین‌ها را تنها بر اساس هیکل انتخاب نمی‌کند، در واقع درست‌تر آن است که گفته شود طبیعت مناسب‌ترین‌ها را انتخاب می‌کند نه بهترین‌ها را.

قانون انتخاب طبیعی به این صورت است که تنها گونه‌هایی از یک جمعیت ادامه نسل می‌دهند که بهترین خصوصیات را داشته باشند و آن‌هایی که این خصوصیات را نداشته باشند به تدریج و در طی زمان از بین می‌روند. به طور مثال فرض کنید گونه خاصی از افراد، هوش بسیار بیشتری از بقیه افراد یک جامعه دارند. در شرایط به طور کامل طبیعی این افراد پیشرفت بهتری خواهند کرد و رفاه به نسبت بالاتری خواهند داشت و این رفاه خود باعث طول عمر بیشتر و باروری بهتر خواهد بود. حال اگر این خصوصیت (هوش) ارثی باشد به طبع در نسل بعدی همان جامعه تعداد افراد باهوش به دلیل زاد و ولد بیشتر این‌گونه افراد بیشتر خواهد بود. اگر همین روند ادامه پیدا کند، ملاحظه می‌شود که در طی نسل‌های متوالی به طور دایم جامعه نمونه، باهوش و باهوش‌تر می‌شود. به این ترتیب یک مکانیزم ساده

---

[1] evolutionary algorithms
[2] generation

طبیعی توانسته است در طی چند نسل افراد کم هوش را از جامعه حذف کند علاوه بر این که میزان هوش متوسط جامعه نیز به طور دایم در حال افزایش است. بدین ترتیب می‌توان دید که طبیعت با بهره‌گیری از یک روش بسیار ساده (حذف تدریجی گونه‌های نامناسب و در عین حال تکثیر بالاتر گونه‌های بهینه) توانسته است به طور دایم هر نسل را از لحاظ خصوصیات مختلف ارتقا بخشد. البته آن‌چه که در بالا ذکر شد به تنهایی توصیف کننده رخدادهای واقعی در تکامل طبیعی نیست. بهینه‌سازی و تکامل تدریجی به خودی خود نمی‌تواند طبیعت را در دسترسی به بهترین نمونه‌ها یاری دهد. این مساله در زیر با یک مثال شرح داده شده است.

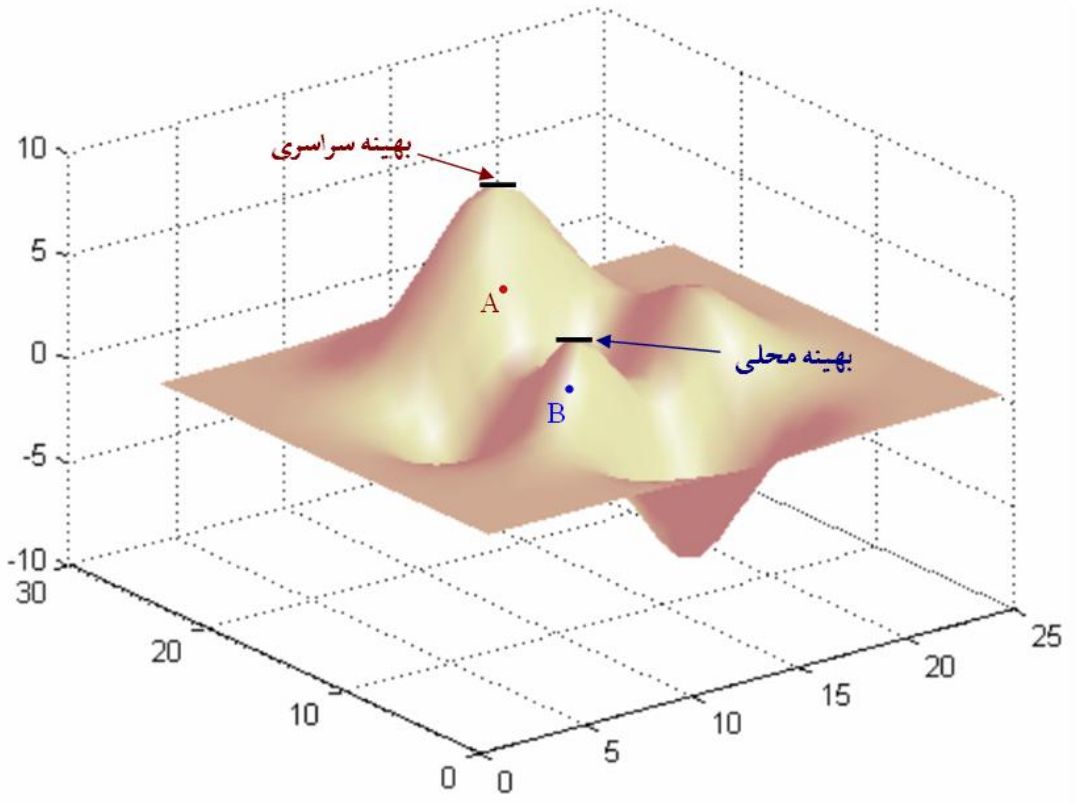
پس از اختراع اتومبیل به تدریج و در طی سال‌ها اتومبیل‌های بهتری با سرعت‌های بالاتر و قابلیت‌های بیشتر نسبت به نمونه‌های اولیه تولید شدند. طبیعی است که این نمونه‌های متاخر حاصل تلاش مهندسان طراح جهت بهینه‌سازی طراحی‌های قبلی بوده‌اند. اما دقت کنید که بهینه‌سازی یک اتومبیل تنها یک «اتومبیل بهتر» را نتیجه می‌دهد. اما آیا می‌توان گفت اختراع هواپیما نتیجه همین تلاش بوده است؟ یا به فرض می‌توان گفت فضا پیماها حاصل بهینه‌سازی طرح اولیه هواپیماها بوده‌اند؟ پاسخ این است که گرچه اختراع هواپیما به طور قطع تحت تاثیر دستاورهای صنعت اتومبیل بوده است اما به هیچ وجه نمی‌توان گفت که هواپیما فقط حاصل بهینه‌سازی اتومبیل و یا فضاپیما حاصل بهینه‌سازی هواپیما است. در طبیعت هم به طور عین همین روند حکم‌فرماست. گونه‌های متکامل‌تری وجود دارند که نمی‌توان گفت فقط حاصل تکامل تدریجی گونه قبلی هستند. در این میان آن‌چه شاید بتواند تا حدودی ما را در فهم این مساله یاری کند مفهومی است به نام *تصادف* یا *جهش*. به عبارتی طرح هواپیما نسبت به طرح اتومبیل یک جهش بود و نه یک حرکت تدریجی. در طبیعت نیز به همین گونه است. در هر نسل جدید بعضی از خصوصیات به صورتی به طور کامل تصادفی تغییر می‌یابند سپس بر اثر تکامل تدریجی که پیش‌تر توضیح داده شد، در صورتی که این خصوصیت تصادفی شرایط طبیعت را ارضا کند حفظ می‌شود در غیر این‌صورت به شکل خودکار از چرخه طبیعت حذف می‌گردد. در واقع می‌توان تکامل طبیعی را به صورت «جستجوی کورکورانه[۱] + بقای قوی‌تر» خلاصه کرد.

حال بررسی می‌شود که رابطه تکامل طبیعی با روش‌های هوش مصنوعی چیست؟ هدف اصلی روش‌های هوشمند به کار گرفته شده در هوش مصنوعی یافتن پاسخ بهینه مسایل مهندسی است. به عنوان مثال، این که چگونه یک موتور طراحی می‌شود تا بهترین بازدهی را داشته باشد یا چگونه بازوهای یک ربات محرک می‌کند تا کوتاه‌ترین مسیر را تا مقصد طی کند (دقت شود که در صورت وجود مانع، یافتن کوتاه‌ترین مسیر دیگر به سادگی کشیدن یک خط راست بین مبدا و مقصد نیست) همگی مسایل بهینه‌سازی هستند. روش‌های کلاسیک ریاضیات دارای دو مشکل اساسی هستند. اغلب این روش‌ها نقطه بهینه محلی[۲] را به عنوان نقطه بهینه سراسری در نظر می‌گیرند و نیز هر یک از این روش‌ها تنها برای مساله خاصی کاربرد دارند. این دو نکته در زیر با مثال‌های ساده‌ای روشن شده است.

---

۱ blind search
۲ local optimal

به شکل ۲-۱ توجه کنید. این منحنی دارای دو نقطه بیشینه می‌باشد. که یکی از آن‌ها تنها بیشینه محلی است. حال اگر از روش‌های بهینه‌سازی ریاضی استفاده شود، به اجبار باید در یک بازه زمانی بسیار کوچک مقدار بیشینه تابع پیدا شود. به طور مثال از نقطه B شروع شده و تابع بیشینه می‌شود. بدیهی است اگر از نقطه B شروع شود، تنها به مقدار بیشینه محلی می‌رسد و الگوریتم پس از آن متوقف خواهد شد. اما در روش‌های هوشمند، همانند الگوریتم ژنتیک (GA)[۱] به دلیل خصوصیت‌های تصادفی آن‌ها حتی اگر هم از نقطه B شروع شود باز هم ممکن است در میان راه، نقطه A به صورت تصادفی انتخاب شود که در این صورت شانس دست‌یابی به نقطه بهینه سراسری[۲] وجود خواهد داشت. در مورد نکته دوم باید گفته شود که روش‌های بهینه‌سازی ریاضی اغلب منجر به یک فرمول یا دستورالعمل خاص برای حل هر مساله می‌شوند، در حالی که روش‌های هوشمند دستورالعمل‌هایی هستند که به صورت کلی می‌توانند در حل بیشتر مساله‌ها به کار گرفته شوند.



شکل ۲-۱: مثالی از نقاط بهینه محلی و بهینه سراسری یک تابع

[۱] Genetic Algorithm
[۲] global optimal

۲-۲ الگوریتم ژنتیک* چیست؟

الگوریتم‌های ژنتیک از اصول انتخاب طبیعی داروین برای یافتن فرمول بهینه جهت پیش بینی یا تطبیق الگو استفاده می‌کنند. الگوریتم‌های ژنتیک اغلب گزینه خوبی برای تکنیک‌های پیش‌بینی بر مبنای رگرسیون هستند. برای مثال اگر نوسانات قیمت نفت با استفاده از عوامل خارجی و ارزش رگرسیون خطی ساده، مدل شود، فرمول زیر تولید خواهد شد:

قیمت نفت در زمان t = ضریب ۱ نرخ بهره در زمان t + ضریب ۲ نرخ بیکاری در زمان t + ثابت ۱

سپس از یک معیار برای پیدا کردن بهترین مجموعه ضرایب و ثابت‌ها جهت مدل کردن قیمت نفت استفاده خواهد شد. در این روش دو نکته اساسی وجود دارد: اول این که این روش خطی است و دوم این که به جای این که در میان «فضای پارامترها» جستجو شود، پارامترهای مورد استفاده مشخص می‌شوند. با استفاده از الگوریتم‌های ژنتیک یک اَبَر ابر فرمول یا یک طرح تنظیم می‌شود، که چیزی شبیه «قیمت نفت در زمان t تابعی از حداکثر ۴ متغیر است» را بیان می‌کند. سپس داده‌هایی برای گروهی از متغیرهای مختلف، شاید در حدود ۲۰ متغیر فراهم می‌شود. پس از آن الگوریتم ژنتیک اجرا خواهد شد که بهترین تابع و متغیرها را مورد جستجو قرار می‌دهد. هر فرمولی که از طرح داده شده بالا تبعیت کند؛ فردی از جمعیت فرمول‌های ممکن تلقی می‌شود. خیلی شبیه به این است که گفته شود «آقای X» فردی از جمعیت انسان‌های ممکن است.

متغیرهایی که هر فرمول داده شده را مشخص می‌کنند، به عنوان یک سری از اعداد نشان داده شده‌اند که به اصطلاح گفته می‌شود که DNA[۱] آن فرد را تشکیل داده‌اند. موتور الگوریتم ژنتیک یک جمعیت اولیه‌ای از فرمول را ایجاد می‌کند. هر فرد در برابر مجموعه‌ای از داده‌های مورد آزمایش قرار می‌گیرند و مناسب‌ترین آن‌ها انتخاب می‌شوند. مناسب‌ترین افراد با هم‌دیگر عمل جفت‌گیری (ترکیب عناصر DNA) و تغییر (تغییر تصادفی عناصر DNA) را انجام می‌دهند. مشاهده می‌شود که با گذشت از میان تعداد زیادی از نسل‌ها، الگوریتم ژنتیک به سمت ایجاد فرمول‌هایی که بیشتر دقیق هستند، میل می‌کنند.

جذابیت زیاد الگوریتم‌های ژنتیک این است نتایج نهایی قابل ملاحظه‌تری دارند. فرمول نهایی برای کاربر قابل مشاهده خواهد بود و برای ارایه سطح اطمینان نتایج می‌توان تکنیک‌های آماری متعارف را بر روی این فرمول‌ها اعمال کرد. به اختصار گفته می‌شود که الگوریتم ژنتیک یک فن برنامه‌نویسی است که از تکامل ژنتیکی به عنوان یک الگوی حل مساله استفاده می‌کند. الگوریتم ژنتیک یک فن جستجو در علوم مهندسی برای یافتن راه حل بهینه ومسایل جستجو است. الگوریتم‌های ژنتیک یکی از انواع

---

*الگوریتم‌های ژنتیک دسته خاصی از الگوریتم‌های تکاملی محسوب می‌شوند که در آن اکثر اوقات از کدگذاری 0 و 1 استفاده می‌شود. به همین دلیل درست آن است که برای کدگذاری‌های دیگر از اصطلاح الگوریتم‌های تکاملی استفاده شود ولی به اشتباه در اکثر متون عنوان الگوریتم ژنتیک به کار می‌رود. برای جلوگیری از ابهام در ذهن خواننده در این فصل در واقع هرجا از اصطلاح ژنتیک استفاده شده منظور همان الگوریتم تکاملی بوده و به دسته خاصی اطلاق نشده است.

[۱] DeoxyriboNucleic Acid

الگوریتم‌های تکاملی هستند؛ که از علم زیست شناسی مثل وراثت[۱]، جهش[۲]، انتخاب ناگهانی، انتخاب طبیعی و ترکیب الهام گرفته شده است.

به طور معمول راه‌حل‌ها به صورت اعداد دودویی نشان داده می‌شوند، ولی روش‌های نمایش دیگری هم وجود دارد که در ادامه این فصل معرفی خواهند شد. تکامل از یک مجموعه به طور کامل تصادفی از موجودیت‌ها شروع می‌شود و در نسل‌های بعدی تکرار می‌شود و در هر نسل، مناسب‌ترین‌ها انتخاب می‌شوند نه بهترین‌ها. راه‌حل‌های هر مساله به وسیله یک لیست از پارامترها نشان داده می‌شود که به آن‌ها کروموزوم[۳] یا ژنوم گفته می‌شود. کروموزوم‌ها به طور معمول به صورت یک رشته[۴] ساده از داده‌ها نمایش داده می‌شوند، البته انواع ساختمان داده‌های دیگر هم می‌توانند مورد استفاده قرار بگیرند. الگوریتم‌های ژنتیک پس از طی چندین مرحله از تولید نسل در نهایت باید بر اساس یک سری شرایط خاصی به اجرای خود خاتمه دهند. تعدادی از شرایط خاتمه الگوریتم‌های ژنتیک عبارتند از:

- به اندازه تعداد محدود و مشخص عمل تولید نسل را انجام دهد.
- به اندازه یک زمان ثابت تعریف شده عمل تولید نسل را انجام دهد.
- یک فرد (فرزند تولید شده) پیدا شود که نتیجه مطلوب را حاصل سازد.
- بیشترین درجه برازندگی فرزندان حاصل شود و یا نتایج بهتری دیگر حاصل نشود.
- با بازرسی دستی متوقف شود.
- به صورت ترکیبی از روش‌های بالا عمل توقف صورت گیرد.

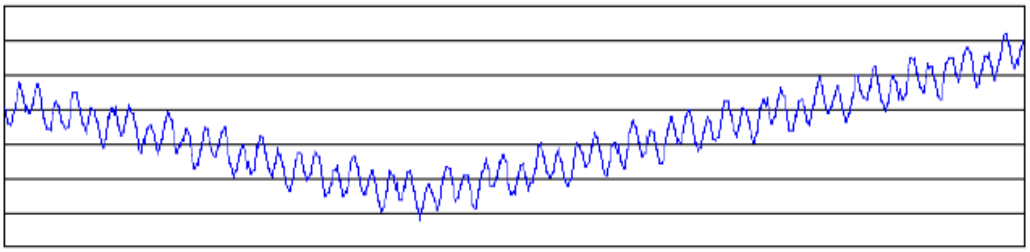
۲-۳ فضای جستجو

موقعی که به جستجوی جواب مساله‌ای پرداخته می‌شود، به طور معمول در میان همه جواب‌های ممکن، جواب‌های بهتر انتخاب می‌شوند. فضای تمام جواب‌های قابل قبول، فضای جستجو نامیده می‌شود. هر نقطه در فضای جستجو یک جواب قابل قبول را نمایش می‌دهد. هر جواب قابل قبول می‌تواند بر اساس ارزش و یا مقدار برازندگی[۵] برای مساله مشخص شود. هدف از پیدا کردن جواب که می‌تواند یک نقطه یا بیشتر در میان جواب‌های قابل قبول باشد، پیدا کردن یک نقطه یا بیشتر در فضای جستجو است. جستجو برای یک جواب، معادل جستجو برای حدود نهایی (حداقل[۶] یا حداکثر[۱]) در

---

۱ inheritance
۲ mutation
۳ chromosome
۴ string
۵ fitness value
۶ minimum

فضای جواب است. کل فضای جستجو از طریق زمان حل یک مساله قابل شناسایی است، اما به طور معمول نقاط کمی از آن فضا مشخص است و از طریق ایجاد نقاط دیگر جواب‌ها به دست آمده‌اند. شکل ۲-۲ نمونه‌ای از فضای جستجو را نمایش می‌دهد [۱۲].



شکل ۲-۲: نمونه‌ای از فضای جستجو [۱۲].

مشکلی که در این جا وجود دارد این است که جستجو می‌تواند خیلی پیچیده باشد. مشخص نیست که کجاها باید به دنبال جواب گشت و در اصل از کجا باید شروع کرد. روش‌های زیادی برای پیدا کردن جواب مناسب (نه فقط بهترین) وجود داردکه به عنوان مثال می‌توان به روش‌های تپه نوردی[۲]، جستجوی محدود، شبیه‌ساز سرد کردن فلزات[۳] و الگوریتم ژنتیک اشاره کرد. جواب‌هایی که از این طریق به دست می‌آیند، به طور معمول جواب‌های خوبی هستند، چون در واقع اثباتی برای این که کدام یکی بهینه واقعی است وجود ندارد.

۲-۴ مسایل NP[۴]

مسایل غیر چند جمله‌ای (NP) نمونه‌هایی از مسایل سخت می‌باشند که از طریق روش‌های سنتی قابل حل نیستند. برای شناخت الگوریتم‌های سریع یا چند جمله‌ای مراحل زیادی باید سپری شود و از طرفی مسایلی هست که به صورت الگوریتمی قابل حل نیستند. برای برخی مسایل ثابت شده است که حل آن‌ها در یک زمان چند جمله‌ای امکان پذیر نیست. البته وقتی یک جواب وجود ندارد، پیدا کردن آن خیلی سخت‌تر است. اگر جواب وجود داشته باشد در این صورت بررسی کردن جواب، کار بسیار ساده‌ای می‌باشد. این مطلب منجر به مسایل NP-Complete می‌شود. حالت NP-Complete بیشتر مربوط به مسایل تصمیم‌گیری است. NP به معنای یک چند جمله‌ای غیر قطعی است و این به این معنا است که می‌توان به وسیله الگوریتم‌های غیرقطعی در یک زمان NP جواب را حدس زد و سپس آن را بررسی نمود. اگر یک ماشین حدس‌زن وجود داشته باشد، آنگاه جوا ب مورد نظر در یک زمان قابل قبولی پیدا می‌شود [۸].

---

۱ maximum
۲ hill climbing
۳ simulated annealing
۴ Non - Polynomial

مطالعه و بررسی مسایل NP-Complete به خاطر سادگی اعمال شده به مساله می‌باشد، چون جواب می‌تواند بلی یا خیر باشد. به خاطر وجود کارها و مسایلی با خروجی‌های بسیار پیچیده، یک گروه از مسایل، مسایل NP-Hard نامیده می‌شوند. این گروه از مسایل به محدودیت گروه مسایل NP-Complete نیستند. حالت NP-Hard بیشتر مربوط به مسایل بهینه‌سازی است. یکی از خصوصیات بارز مسایل NP این است که هنگام رویارویی با این مسایل، الگوریتم یا الگوریتم‌هایی را که برای حل این مسایل مناسب هستند به راحتی می‌توان یافت و کار آن‌ها تنها جستجوی تمام جواب‌های ممکن است. اما مشکل این است که این الگوریتم‌ها بسیار کند هستند (به طور معمول $O(2^n)$) و حتی گاهی اوقات برای یک بیت اضافه‌تر به منظور ایجاد نمونه‌های بزرگ‌تر، الگوریتم غیر قابل استفاده می‌شود.

امروزه کسی نمی‌داند که آیا الگوریتم‌های دقیق سریع‌تری برای حل مسایل NP هم وجود دارد یا خیر؟ اثبات یا عدم اثبات این مطلب جزو وظایف محققان امروزی می‌باشد. امروزه خیلی از محققین بر این باور هستند که چنین الگوریتمی وجود ندارد و بنابراین به دنبال یافتن بعضی روش‌های جایگزین یا فرعی هستند که الگوریتم ژنتیک یکی از این روش‌ها می‌باشد.

۲-۵ مفاهیم اولیه در الگوریتم ژنتیک

قبل از ارایه کردن الگوریتم ژنتیک لازم است مفاهیم اولیه این روش و اجزای اصلی تشکیل دهنده آن بررسی شوند، که در زیر به ترتیب به شرح آن‌ها پرداخته شده است [۱۳]:

۲-۵-۱ اصول پایه‌ای

الگوریتم‌های ژنتیکی بر اساس نظریه تکاملی داروین می‌باشند و جواب مساله‌ای که از طریق الگوریتم ژنتیک حل می‌شود رفته رفته بهبود می‌یابد. الگوریتم ژنتیک با یک مجموعه از جواب‌ها که از طریق کروموزوم‌ها نشان داده می‌شوند شروع می‌شود. این مجموعه جواب‌ها جمعیت[۱] اولیه نام دارد. در این الگوریتم جواب‌های حاصل از یک جمعیت برای تولید جمعیت بعدی استفاده می‌شوند. در این فرآیند امید است که جمعیت جدید نسبت به جمعیت قبلی بهتر باشد. انتخاب بعضی از جواب‌ها از میان کل جواب‌ها (والدین)[۲] به منظور ایجاد جواب‌های جدید یا همان فرزندان[۳] بر اساس میزان برازندگی آن‌ها می‌باشد. طبیعی است که جواب‌های مناسب‌تر شانس بیشتری برای تولید مجدد داشته باشند. این فرآیند تا برقراری شرطی که تعیین شده است (مانند تعداد جمعیت‌ها یا میزان بهبود جواب) ادامه پیدا می‌کند.

---

۱ population
۲ parents
۳ offspring

۲-۵-۱-۱ نمای کلی الگوریتم ژنتیک

در شکل ۲-۳ نمای کلی الگوریتم ژنتیک نمایش داده شده است. همان طور که مشاهده می‌شود، اصول پایه‌ای الگوریتم ژنتیک بسیار عمومی است. بنابراین برای مسایل مختلف فاکتورهای مختلف زیادی وجود دارد که باید مورد بررسی قرار گیرد. اولین سوال این است که ایجاد یک کروموزوم چگونه است؟ یا این که چه نوعی از کدگذاری انتخاب شود؟

دو عمل‌گر بسیار مهم و پایه‌ای الگوریتم ژنتیک عمل‌گرهای ترکیب و جهش می‌باشند. سوال بعدی این است که برای ترکیب والدین به منظور ایجاد فرزندان جدید والدین چگونه انتخاب شوند؟ این کار به روش‌های مختلفی می‌تواند صورت گیرد. اما ایده اصلی در تمام آن‌ها این است که والدین بهتر انتخاب شوند به این امید که والدین بهتر باعث ایجاد فرزندان بهتر شوند. مساله‌ای که در این‌جا مورد سوال می‌باشد این است که اگر جمعیت جدید تنها از طریق فرزندان جدید ایجاد شود، این فرآیند منجر به حذف کروموزوم‌های نسل قبل می‌گردد. برای جلوگیری از این پیشامد، همیشه بهترین جواب نسل قبل بدون هیچ تغییری به نسل جدید منتقل می‌شود.

۱. تولید جمعیت اولیه[۱] شامل n کروموزوم
۲. بررسی تابع ارزیابی $f(x)$ برای هر کروموروم x در جمعیت
۳. ایجاد یک جمعیت جدید بر اساس تکرار قدم‌های زیر:
  ۳-۱. انتخاب دو کروموزوم والد از یک جمعیت بر اساس میزان برازندگی آن‌ها
  ۳-۲. در نظر گرفتن مقدار مشخصی برای احتمال اعمال عمل‌گر ترکیب[۲] (تقاطع) و سپس انجام عملیات ترکیب بر روی والدین به منظور ایجاد فرزندان (اگر هیچ ترکیب جدیدی صورت نگیرد، فرزندان همان والدین خواهند بود)
  ۳-۳. در نظر گرفتن احتمال جهش و سپس تغییر فرزندان در هر مکان
  ۳-۴. جایگزین فرزندان جدید در جمعیت جدید
۴. استفاده از جمعیت جدید برای اجراهای بعدی الگوریتم
۵. توقف اجرای الگوریتم در صورت مشاهده شرایط توقف و برگرداندن بهترین جواب در جمعیت فعلی
۶. رفتن به مرحله ۲

شکل ۲-۳: نمای کلی الگوریتم ژنتیک

[۱] Initial population
[۲] crossover

۲-۵-۲ کدگذاری[۱]

الگوریتم ژنتیک به جای این که بر روی پارامترها یا متغیرهای مساله کار کند، با شکل کد شده آن‌ها سر و کار دارد. روش‌های کدگذاری متداول در الگوریتم ژنتیک عبارتند از:

۱. کدگذاری دودویی[۲]
۲. کدگذاری جهشی[۳]
۳. کدگذاری ارزشی[۴]
۴. کدگذاری درختی[۵]

تعداد بیت‌هایی که برای کدگذاری متغیرها استفاده می‌شود وابسته به دقت مورد نظر برای جواب‌ها، محدوده تغییرات پارامترها و رابطه بین متغیرها می‌باشد. کدگذاری در حالت کلی به دو صورت زیر انجام می‌گیرد:

۱. کدگذاری مستقیم: در این روش کل یک جواب به عنوان یک کروموزوم در نظر گرفته می‌شود. برای مسایل پیچیده چنین روشی مناسب نیست؛ زیرا عمل‌گرهای ژنتیکی به خاطر گستردگی زیاد فرزندان، غیر کاربردی می‌شوند و در نتیجه منجر به جواب‌های غیر قابل قبول و قانونی می‌شوند.

۲. کدگذاری غیر مستقیم: در این روش تنها قسمتی از یک جواب به صورت یک کروموزوم کدگذاری می‌شود.

حال در زیر هر چهار روش کدگذاری که در بالا معرفی شدند، توضیح داده می‌شود.

۲-۵-۲-۱ کدگذاری دودویی

این کدگذاری، متداول‌ترین نوع کدگذاری است. در این روش، هر کروموزوم یک رشته از بیت‌های شامل 0 و 1 می‌باشد. کدگذاری دودویی می‌تواند حالت‌های زیادی را پوشش دهد. در شکل ۲-۴ مثالی برای کدگذاری دودویی نمایش داده شده است.

کروموزوم A	1	0	1	0	1	0	0	0	0
کروموزوم B	0	1	1	0	0	1	0	1	1

شکل ۲-۴: مثالی ازکدگذاری دودویی

---

۱ encoding
۲ binary encoding
۳ permutation encoding
۴ value encoding
۵ tree encoding

از طرف دیگر این نوع کدگذاری برای خیلی از مسایل حالت طبیعی ندارد و اغلب اوقات لازم است که بعد از عمل‌گرهای ترکیب و جهش اصلاحاتی صورت بگیرد.

۲-۵-۲-۲ کدگذاری جهشی

این نوع کدگذاری می‌تواند در مسایل ترتیبی نظیر مساله فروشنده دوره گرد (TSP[۱]) یا مساله زمان‌بندی کارها (JS[۲]) به کار رود. در کدگذاری جهشی، هر کروموزوم یک رشته از اعداد می‌باشد. شکل ۲-۵ نمونه‌ای از این نوع کدگذاری را نمایش می‌دهد.

کروموزوم A	1	5	3	2	6	4	7	9	8
کروموزوم B	8	5	6	7	2	3	1	4	9

شکل ۲-۵: مثالی ازکدگذاری جهشی

کدگذاری به روش جهشی تنها برای مسایل ترتیبی مفید است. حتی برای همین مسایل نیز گاهی اوقات باید عمل‌گرهای ترکیب و جهش به صورت اصلاح شده برای ایجاد کروموزوم‌های سازگار و مناسب انجام گیرد.

۲-۵-۲-۳ کدگذاری ارزشی

این نوع کدگذاری در مسایلی که در آن‌ها مقادیر پیچیده نظیر اعداد حقیقی به کار می‌روند استفاده می‌شود. استفاده از این نوع کدگذاری برای چنین مسایلی بسیار سخت می‌باشد. در کدگذاری ارزشی هر ژن[۳] در یک کروموزوم ارزش خاصی دارد. این پارامتر با ارزش می‌تواند عدد، حرف و یا کلمه باشد. در این نوع کدگذاری نیاز به توسعه عمل‌گرهای جابه‌جایی و جهش جدیدی برای مسایل خاص می‌باشد. در شکل ۲-۶ مثالی از کدگذاری ارزشی نمایش داده شده است.

کروموزوم A	1,2324	5,3243	0,4556	2,3293	2,4545
کروموزوم B	ABDJE	IFJDH	DIERJ	FDLDF	LFEGT
کروموزوم C	(Back)	(Back)	(Right)	(Forward)	(Left)

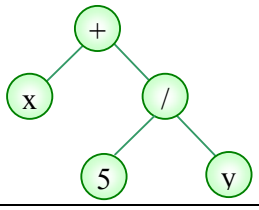
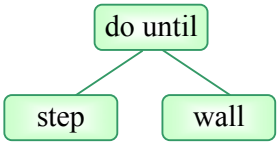
شکل ۲-۶: مثالی ازکدگذاری ارزشی

---

۱ Traveling Salesman Problem
۲ Job Scheduling
۳ gene

۲-۵-۲-۴ کدگذاری درختی

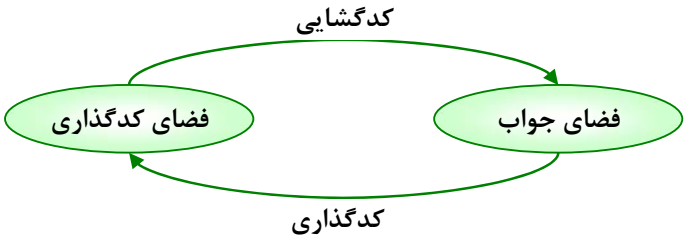
کدگذاری درختی در برنامه‌های تکاملی به منظور برنامه‌ریزی تکاملی به کار می‌رود. در کدگذاری درختی هر کروموزوم یک درخت از اشیایی نظیر توابع یا دستورها در زبان برنامه‌نویسی می‌باشد. شکل ۲-۷ دو نمونه از این کروموزوم‌ها را نمایش می‌دهد. این نوع کدگذاری برای برنامه‌های تکاملی بسیار خوب است. زبان برنامه‌نویسی LISP اغلب از این نوع کدگذاری استفاده می‌کند و آن به این دلیل است که برنامه‌های آن به این فرم نمایش داده می‌شوند و می‌توانند به راحتی مورد تجزیه قرار بگیرند. بنابراین عمل‌گرهای ترکیب و جهش نیز به همان نسبت راحت انجام می‌شوند.

(+ x (/ 5 y))	(انجام تا برقراری شرط خاتمه)
کروموزوم A	کروموزوم B
	

شکل ۲-۷: مثالی ازکدگذاری درختی [۱۲].

۲-۵-۲-۵ مسایل مربوط به کدگذاری

نکته‌ای که در انتهای این بخش باید به آن توجه کرد این است که در الگوریتم‌های ژنتیک کدگذاری یک رابطه بین فضای کدگذاری و فضای جواب‌ها می‌باشد، به طوری که الگوریتم ژنتیک عملیات تکاملی را به طور متناوب در این دو فضا انجام می‌دهد (شکل ۲-۸). انتخاب طبیعی نیز به عنوان یک رابطه بین کروموزوم‌ها و عملکرد جواب‌های کد شده آن‌ها می‌باشد.



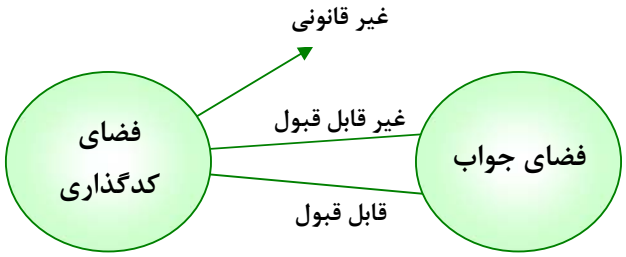
شکل ۲-۸: فضای کدگذاری و فضای جواب [۱۲].

همان طور که ذکر شد، نحوه کدگذاری یک جواب به صورت یک کروموزوم یک موضوع کلیدی در الگوریتم ژنتیک می‌باشد. در کدگذاری‌های غیر رشته‌ای سه موضوع بسیار مهم مطرح است که عبارتند از:

- قابل قبول بودن[۱] کروموزوم
- قانونی بودن[۲] کروموزوم
- یگانی رابطه

قابل قبول بودن یک کروموزوم به این مفهوم است که آیا کدگشایی[۳] این کروموزوم در ناحیه قابل قبول که جزئی از فضای جواب مساله است قرار دارد یا خیر؟ غیر قابل قبول بودن یک کروموزوم ناشی از طبیعت مسایل بهینه‌سازی با محدودیت می‌باشد. به طور معمول در مسایل بهینه‌سازی فضای قابل قبول به وسیله یک سامانه معادلات یا نامعادلات تشکیل می‌شود. در چنین مواردی روش‌های جریمه[۴] کردن متعددی به منظور جلوگیری از ایجاد کروموزوم‌های غیر قابل قبول پیشنهاد شده است. در مسایل بهینه‌سازی با محدودیت به طور معمول جواب در مرز بین فضای قابل قبول و فضای غیر قابل قبول قرار دارد. در نتیجه روش‌های جریمه کردن الگوریتم ژنتیک را مجبور می‌کنند که از هر دو طرف به سمت جواب حرکت کند.

قانونی بودن یک کروموزوم به این مفهوم است که آیا کروموزوم در اثر کدگشایی به یک جواب منجر خواهد شد یا خیر؟ یعنی در فضای جواب قرار خواهد گرفت یا خیر؟ غیر قانونی بودن کروموزوم ناشی از طبیعت روش‌های کدگذاری می‌باشد. چون یک کروموزوم غیرقانونی قادر نیست به یک جواب تبدیل شود، در نتیجه ارزیابی چنین کروموزومی غیر ممکن است و بنابراین روش‌های جریمه‌ای در این‌جا عملی نیستند. در چنین مواردی به طور معمول از روش‌های ترمیم[۵] به منظور تبدیل کروموزوم‌های غیر قانونی به کروموزوم‌های قانونی استفاده می‌کنند. در شکل ۲-۹ رابطه بین کروموزوم‌ها و جواب‌ها نمایش داده شده است.



شکل ۲-۹: رابطه بین کروموزوم‌ها و جواب‌ها [۱۲].

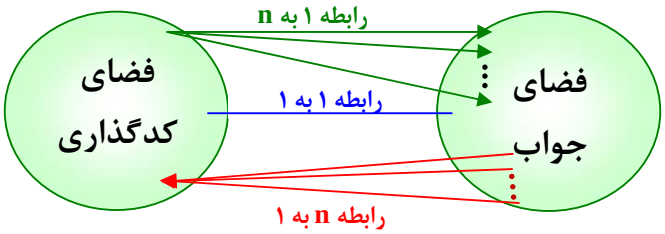
مساله سوم، بحث یکسانی رابطه است. روابط بین کروموزوم‌ها و جواب‌ها به طور معمول به یکی از سه صورت زیر است:

---

۱ feasible
۲ legality
۳ decoding
۴ penalty methods
۵ repairing techniques

- ❖ رابطه یک به یک
- ❖ رابطه n به یک
- ❖ رابطه n به n

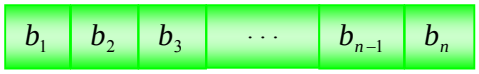
شکل ۲-۱۰ این سه نوع رابطه را نمایش می‌دهد. از این سه نوع رابطه، رابطه یک به یک بهترین نوع رابطه و رابطه n به n بدترین نوع آن است.



شکل ۲-۱۰: انواع جواب بین فضای جواب و فضای کدگذاری [۱۲].

۲-۵-۳ کروموزوم

رشته یا دنباله‌ای از بیت‌ها که به عنوان شکل کد شده یک جواب ممکن (مناسب یا نامناسب) از مساله مورد نظر می‌باشد را کروموزوم گویند. در حقیقت بیت‌های یک کروموزوم، نقش ژن‌ها را در طبیعت بازی می‌کنند. هر بیت متغیر گسسته است که از یک مجموعه Q عضوی، انتخاب می‌شود. چنان‌چه از کدگذاری دودویی استفاده شود، هر بیت یکی از دو مقدار 0 و یا 1 را خواهد پذیرفت، بنابراین $Q = 2$ می‌باشد. یک کروموزوم دارای n ژن یا بیت می‌باشد که در شکل ۲-۱۱ نمایش داده شده است. b_i در این شکل نشان دهنده مقدار بیت i ام است که از مجموعه $Q = m$ عضوی انتخاب می‌شود.



شکل ۲-۱۱: نمایش یک کروموزوم n بیتی در مبنای عددی m

$$b_i \in (0, 1, ..., m) \qquad i = 1, 2, ..., n$$

۲-۵-۴ جمعیت ژنتیکی

به مجموعه‌ای از کروموزوم‌ها جمعیت گفته می‌شود. یکی از ویژگی‌های الگوریتم‌های ژنتیک این است که به جای تمرکز بر روی یک نقطه از فضای جستجو یا یک کروموزوم، بر روی جمعیتی از کروموزوم‌ها کار می‌کنند. به این ترتیب در هر مرحله، الگوریتم دارای جمعیتی از کروموزوم‌ها بوده که خواص مورد نظر را بیشتر از جمعیت مرحله قبل دارا می‌باشد. هر جمعیت یا یک نسل از کروموزوم‌ها، دارای یک

اندازه می‌باشد که به اندازه جمعیت[۱] معروف است. اندازه جمعیت معرف تعداد کروموزوم‌های موجود در جمعیت یا یک نسل است. اگر تعداد کروموزوم‌ها خیلی کم باشد، امکان شکل‌گیری عملیات جابه‌جایی به وسیله الگوریتم ژنتیک بسیار کم خواهد بود و تنها قسمت کمی از فضای جستجو مورد کاوش قرار خواهد گرفت. از طرف دیگر، اگر تعداد کروموزوم‌ها خیلی زیاد باشد، سرعت الگوریتم بسیار کند خواهد شد. بر اساس تحقیقات، جمعیت‌های با اندازه مناسب حدود ۲۰ تا ۳۰ کروموزوم دارند. البته گاهی اوقات جمعیت با اندازه ۵۰ تا ۱۰۰ بهترین جواب‌ها را داده‌اند. بعضی از تحقیقات نیز نشان می‌دهد که اندازه جمعیت باید بر اساس نوع مساله و کدگذاری آن تعریف شود و افزایش بیشتر آن بی فایده خواهد بود و هرگز به حل سریع‌تر مساله کمک نمی‌کند.

۲-۶ تابع برازندگی

یکی از مراحل الگوریتم ژنتیک ارزیابی جواب‌های به دست آمده در هر مرحله است. در واقع ارزش جواب‌های به دست آمده در هر مرحله تعیین می‌شود. مناسب بودن یا نبودن جواب با معیاری که از تابع هدف به دست می‌آید، سنجیده می‌شود. هرچه که یک جواب مناسب‌تر باشد مقدار برازندگی بیشتری دارد که با استفاده از محدوده دانش مساله به کار برده می‌شود. برای آن که شانس بقای چنین جوابی بیشتر شود احتمال بقای آن متناسب با مقدار برازندگی آن در نظر گرفته می‌شود. بنابراین رشته‌ای که برازنده‌تر است با احتمال بیشتری در تولید فرزندان شرکت می‌کند و دنباله‌های بیشتری را به وجود می‌آورد. با ارزش‌ترین جواب‌ها در هر مرحله مانند قوی‌ترین موجودات در یک جمعیت می‌باشند. در تکثیر، رشته‌های با میزان تطبیق کم، از جمعیت حذف می‌شوند و رشته‌های با میزان تطبیق زیاد، تاثیر بیشتری در تولید جمعیت بعدی خواهند داشت. به طور معمول در مواردی که امکان دارد، تابع برازندگی را در فاصله (1-0) نرمالیزه[۲] می‌کنند [۲].

۲-۷ عمل‌گر ترکیب یا جابه‌جایی

ترکیب به معنی باز ترکیب اطلاعات ژنتیکی بین کروموزوم‌ها می‌باشد و در واقع عمل‌گر ترکیب یک روش برای اشتراک اطلاعات مابین کروموزوم‌ها می‌باشد. این عمل‌گر خصیصه‌های والدین را برای ساختن فرزندان ترکیب می‌کند تا این که کروموزوم‌های بهتری ایجاد شوند. به طور معمول عمل‌گر ترکیب روی یک جفت از کروموزوم‌ها عمل می‌کند و دو فرزند برای هر جفت تولید می‌شود. عمل‌گر ترکیب می‌تواند روی چندین والد نیز عمل کند که در این صورت خصیصه‌های بیش از دو والد را برای تولید فرزندان ترکیب می‌کند. علاوه بر این عمل‌گر ترکیب می‌تواند بیش از دو فرزند را برای هر گروه از والدین تولید کند. وظیفه اصلی این عمل‌گر بهبود برازندگی جمعیت می‌باشد. عمل‌گر ترکیب برای حفظ و تنوع و گوناگونی جمعیت اجازه نمی‌دهد که فرزندان فقط ژن‌های خوب را به ارث ببرند. پیاده‌سازی

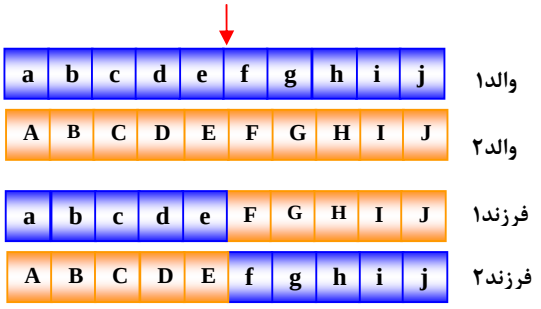
---

[۱] population size
[۲] normalization

این عمل‌گر به روش کدگذاری کروموزوم‌ها وابسته می‌باشد. بر حسب این که ژن‌های والد چگونه ژن‌های فرزندان را تولید می‌کنند عمل‌گر ترکیب به انواع مختلفی تقسیم می‌شوند. در زیر تعدادی از متداول‌ترین روش‌های ترکیب آمده است. در هرکدام از این روش‌ها پس از ایجاد فرزندان از بین کروموزوم‌های والد و کروموزوم‌های فرزند، دو کروموزومی که بیشترین مقدار برازندگی را داشته باشند به نسل بعدی انتقال می‌یابند. نکته‌ای که باید به آن توجه کرد این است که عمل‌گر ترکیب ممکن است فرزند نامعتبری را تولید کند؛ بنابراین پس از اعمال این عمل‌گر باید بررسی شود که فرزندان تولید شده معتبر باشند [۱۵].

۲-۷-۱ ترکیب تک نقطه‌ای (1PX[۱])

این روش متداول‌ترین روش ترکیب می‌باشد. در این روش یک نقطه به طور تصادفی در کروموزوم‌های والد انتخاب شده و با هم‌دیگر ترکیب می‌شوند. مثالی از ترکیب تک نقطه‌ای در شکل ۲-۱۲ نمایش داده شده است.



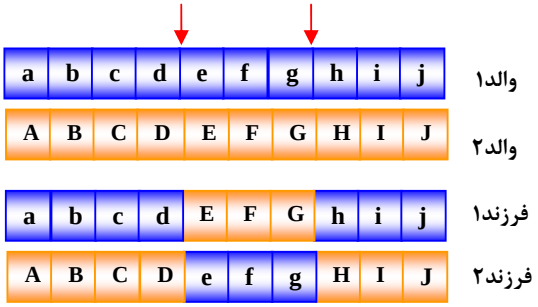
شکل ۲-۱۲: ترکیب تک نقطه‌ای

۲-۷-۲ ترکیب چند نقطه‌ای (NPX[۲])

در این روش n نقطه به طور تصادفی در کروموزوم‌های والد انتخاب شده و سپس ژن‌های بین نقاط انتخاب شده به صورت یک در میان با هم‌دیگر عوض می‌شوند. هرچه تعداد نقاط انتخاب شده زیاد باشد، کارآیی نیز افزایش پیدا می‌کند. تعداد نقاط جابه‌جایی بهینه که مورد نظر قبول عموم قرار گرفته است، به طور معمول نصف طول کروموزوم می‌باشد. روش‌های جابه‌جایی یک نقطه‌ای و چند نقطه‌ای موقعی که کروموزوم‌ها به صورت بیتی کدگذاری شده باشند، مفید می‌باشند. مثالی از ترکیب چند نقطه‌ای در شکل ۲-۱۳ نمایش داده شده است.

---

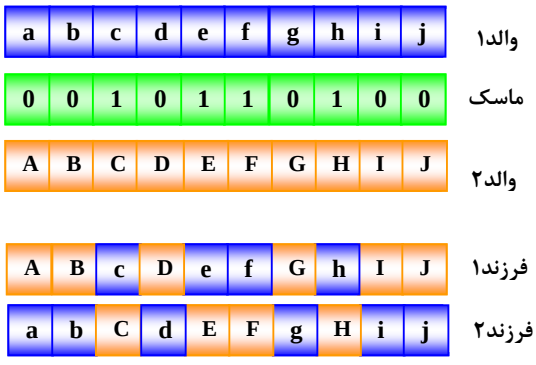
۱ One-Point Crossover
۲ N-Point Crossover



شکل ۲-۱۳: ترکیب چند نقطه‌ای

۲-۷-۳ ترکیب یکنواخت[۱]

در این روش تولید فرزندان از روی نقاب[۲] که به طور تصادفی تولید می‌شود انجام می‌گیرد. به این صورت که اگر در نقاب بیتی برابر ۱ باشد والد اول در تولید فرزند اول و والد دوم در تولید فرزند دوم به‌کار می‌رود. یعنی ژن متناظر از والد اول در فرزند اول و ژن متناظر از والد دوم در فرزند دوم کپی می‌شود و اگر در نقاب بیتی برابر 0 باشد والد اول در تولید فرزند دوم و والد دوم در تولید فرزند اول به کار می‌رود، به عبارتی، ژن متناظر از والد اول در فرزند دوم و ژن متناظر از والد دوم در فرزند اول کپی می‌شود. مثالی از ترکیب یکنواخت در شکل ۲-۱۴ نمایش داده شده است.



شکل ۲-۱۴: ترکیب یکنواخت

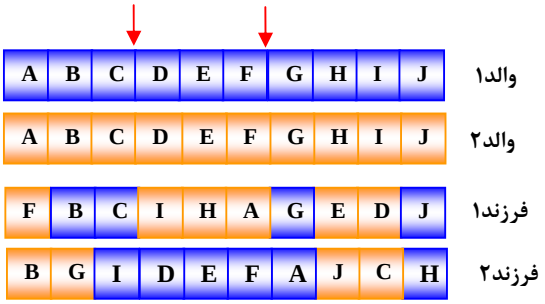
۲-۷-۴ ترکیب نگاشت جزئی (PMX[۳])

مشکل ترکیب‌های تک نقطه‌ای و چند نقطه‌ای این است که برخی از ژن‌ها ممکن است بیش از یک بار در کروموزوم ظاهرشوند و برخی از ژن‌ها در کروموزوم ظاهر نشوند. برای رفع این مشکل از روش PMX استفاده می‌شود. این روش موقعی که کروموزوم‌ها به صورت جایگشتی کدگذاری شده باشند به کار می‌رود. اگر جایگشت‌های والد اول و والد دوم در نظر گرفته شود دو جایگشت فرزند به وسیله انتخاب دو

---

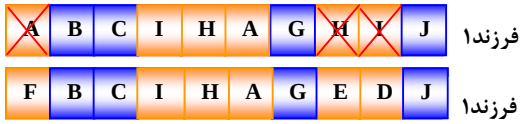
۱ Uniform Crossover
۲ mask
۳ Partially Mapped Crossover

نقطه برش در جایگشت‌های والد و کپی کردن عناصر بین دو نقطه برش در فرزندان و پرکردن باقی‌مانده جایگشت‌های فرزندان با عناصر متناظر آن‌ها از والدین، ایجاد می‌شوند. توجه کنید که اگر یکی از عناصر خارج از ناحیه جابه‌جایی فرزندی با یکی از عناصر ناحیه جابه‌جایی خود یکسان باشد عنصر به عنوان حفره در نظر گرفته می‌شود و باید مقدار آن عوض شود. نحوه پرکردن یک حفره به صورت شکل ۲-۱۶ می‌باشد. به فرض اگر حفره در فرزند اول باشد، ابتدا موقعیت ژنی از والد دوم را که دارای مقدار مساوی با مقدار حفره است پیدا می‌شود. سپس از والد اول مقدار ژنی را که در موقعیت یکسانی با ژن پیدا شده قرار دارد به عنوان مقدار حفره در نظر گرفته می‌شود. مثالی از ترکیب نگاشت جزئی در شکل ۲-۱۵ نمایش داده شده است.



شکل ۲-۱۵: ترکیب نگاشت جزئی

شناسایی حفره‌ها و پر کردن آن‌ها برای فرزند ۱ در شکل ۲-۱۶ نمایش داده شده است.

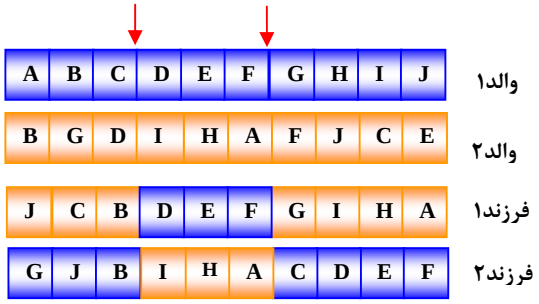


شکل ۲-۱۶: نحوه پر کردن حفره‌ها

۲-۷-۵ ترکیب مرتب شده (OX[1])

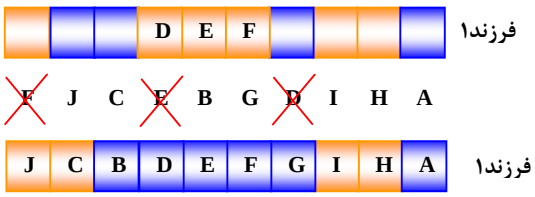
این روش، ترتیب نسبی ژن‌ها در کروموزوم‌ها را تا حد ممکن حفظ می‌کند. جایگشت‌های والد اول و دوم را در نظر بگیرید. دو جایگشت فرزند به وسیله انتخاب دو نقطه برش در جایگشت‌های والد و کپی کردن عناصر بین دو نقطه برش در فرزندان، و پر کردن باقی‌مانده جایگشت‌های فرزندان با عناصر استفاده نشده از والد دیگر با شروع از نقطه برش دوم به بعد، ایجاد می‌شوند. مثالی از ترکیب مرتب در شکل ۲-۱۷ نمایش داده شده است.

[1] Ordered Crossover



شکل ۲-۱۷: ترکیب مرتب

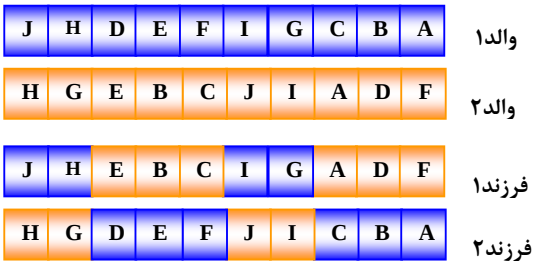
به عنوان مثال برای ایجاد فرزند اول عناصر بین نقاط برش از والد اول کپی می‌شود. سپس باقی‌مانده عناصر فرزند با شروع از نقطه برش دوم والد دوم و حذف عناصری که از قبل در فرزند اول دیده شده‌اند پر می‌شوند. فرزند دوم نیز به همین ترتیب ایجاد می شود.



شکل ۲-۱۸: نحوه پر کردن عناصر

۲-۷-۶ ترکیب چرخشی (CX[1])

جایگشت‌های والدین را در نظر بگیرید، دو جایگشت فرزند به وسیله شکل دادن یک چرخه در بین والدین ایجاد می‌شوند. مثالی از ترکیب دوره‌ای در شکل ۲-۱۹ نمایش داده شده است.

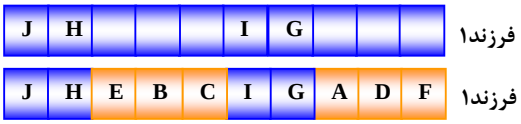


شکل ۲-۱۹: ترکیب دوره‌ای

برای مثال برای ایجاد فرزند اول، با شروع از اولین عنصر والد۱، J دیده می‌شود و آن به H در والد دوم نگاشت می‌شود H در والد۱ به G در والد۲ نگاشت می‌شود. G در والد ۱ به I در والد ۲ نگاشت

[1] Cycle Crossover

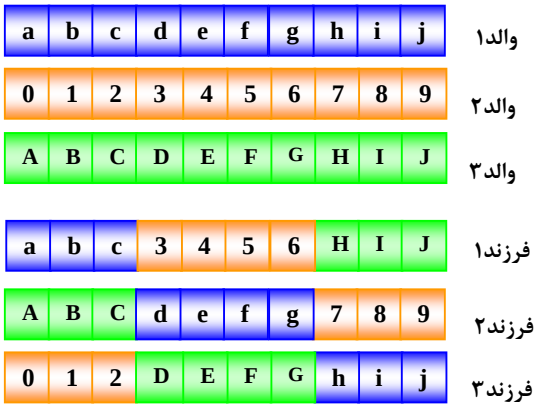
می‌شود. I در والد ۱ به J در والد ۲ نگاشت می‌شود و چرخه کامل می‌شود. عناصری از خرچه که در والد ۱ هستند در فرزند اول قرار داده می‌شود. سپس مکان‌های خالی فرزند اول با عناصر متناظر در والد۲ پر می‌شوند. فرزند دوم نیز به همین ترتیب ایجاد می‌شود.



شکل ۲-۲۰: نحوه پر کردن عناصر

۲-۷-۷ ترکیب مورب (DX[1])

این روش تعمیم یافته روش تک نقطه‌ای می‌باشد. برای n والد با انتخاب n-1 نقطه جابه‌جایی و برداشتن ژن‌های بین نقاط جابه‌جایی کروموزوم‌های والدین در امتداد قطر n کروموزوم فرزند ایجاد می‌شود. این روش برای n = 3 در شکل ۲-۲۱ نشان داده شده است.



شکل ۲-۲۱: ترکیب مورب

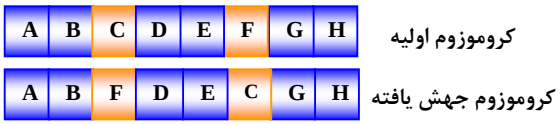
۲-۸ عمل‌گر جهش

بعد از اعمال عمل‌گر ترکیب به منظور اجتناب از هم‌گرایی به بهینه محلی و ایجاد تنوع و گوناگونی در جمعیت با استفاده از عمل‌گر جهش یک تعداد از کروموزوم‌های به دست آمده تغییر داده می‌شوند. با این عمل کروموزوم‌های جدیدی که به احتمال در کل جمعیت وجود نداشته‌اند به وجود می‌آید. در زیر تعدادی از متداول‌ترین روش‌های جهش آمده است [۱۵].

[1] Diagonal Crossover

۲-۸-۱ روش تعویض[۱]

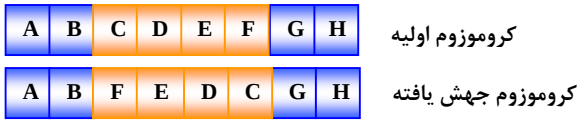
دو ژن یا دو بلوک از ژن‌ها در کروموزوم به طور تصادفی انتخاب شده و موقعیت آن‌ها عوض می‌شود. شکل ۲-۲۲ مثالی از جهش به روش تعویض را نمایش می‌دهد.



شکل ۲-۲۲: روش تعویض

۲-۸-۲ روش وارون سازی[۲]

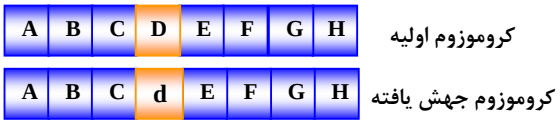
دو ژن از کروموزوم به طور تصادفی انتخاب شده و ترتیب ژن‌ها در بین این دو ژن معکوس می‌شود. شکل ۲-۲۳ مثالی از جهش به روش وارون سازی را نمایش می‌دهد.



شکل ۲-۲۳: روش وارون سازی

۲-۸-۳ روش ژن جزئی[۳]

یک ژن از کروموزوم به طور تصادفی انتخاب شده و مقدار آن را به طور تصادفی عوض می‌شود. شکل ۲-۲۴ مثالی از جهش به روش ژن جزئی را نمایش می‌دهد.



شکل ۲-۲۴: روش ژن جزئی

۲-۸-۴ روش درجی[۴]

یک ژن یا یک بلوک از ژن‌ها در کروموزوم به طور تصادفی انتخاب شده و آن‌ها را در یک نقطه تصادفی دیگر درج می‌شود. شکل ۲-۲۵ مثالی از جهش به روش درجی را نمایش می‌دهد.

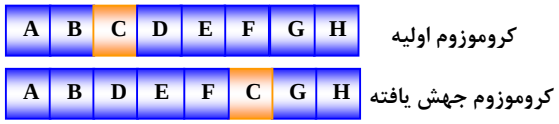
---

[۱] order-based (swap) mutation

[۲] sub list-based(inversion) mutation

[۳] partial-gene mutation

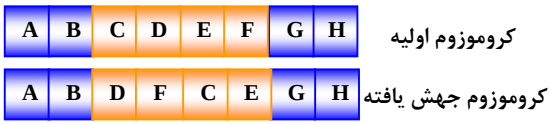
[۴] insertion mutation



شکل ۲-۲۵: روش درجی

۲-۸-۵ روش در هم آمیخته[۱]

یک بلوک از ژن‌ها به طور تصادفی انتخاب شده و آن‌ها به صورت تصادفی دوباره مرتب می‌شوند. شکل ۲-۲۶ مثالی از جهش به روش درهم آمیخته را نمایش می‌دهد.



شکل ۲-۲۶: روش در هم آمیخته

۲-۹ فرآیند انتخاب[۲]

پس از اعمال عمل‌گرهای GA مجموعه‌ای از رشته‌ها برای دور محاسباتی بعد با تعدادی برابر جمعیت اولیه تعیین می‌شوند. این مجموعه تشکیل شده از کلیه رشته‌های فرزند (حاصل از عمل‌گرهای ترکیب و جهش) و تعدادی از رشته‌های مرحله قبل می‌باشد. انتخاب این تعداد از رشته‌های اولیه به طور تصادفی از بین رشته‌های با عدد برازندگی بالاتر انجام می‌شود. این مرحله را تکثیر گویند. در زیر برخی از روش‌های انتخاب معرفی می‌شود [۱۴].

۲-۹-۱ روش چرخ رولت[۳]

روش چرخ رولت یکی از متداول‌ترین روش‌های انتخاب می‌باشد. انتخاب یک شخص می‌تواند، با تعیین احتمال انتخاب برای هر شخص انجام گیرد که این احتمال با نسبت برازندگی آن شخص نسبت به مجموع برازندگی تمامی اشخاص در جمعیت برابر است. توزیع احتمال انتخاب می‌تواند با استفاده از معادله زیر تولید شود.

$$p(k) = \frac{F_k}{\sum_{i=1}^{n} F_i} \qquad (۱-۲)$$

---

۱ scramble mutation
۲ selection mechanism
۳ roulette wheel selection

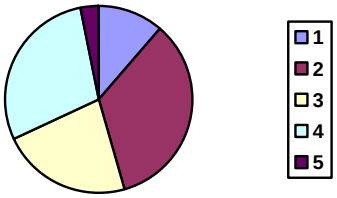
در رابطه بالا $p(k)$ عبارت است از احتمال انتخاب کروموزوم kام، F_k عبارت است از میزان برازندگی کروموزوم kام، $\sum_{i=1}^{n} F_i$ عبارت است از مجموع میزان برازندگی تمام کروموزوم‌ها و n عبارت است از تعداد کروموزوم‌های جمعیت.

عمل انتخاب طبق مراحل زیر صورت می‌گیرد:

۱. برازندگی تمام اعضای جمعیت حساب می‌شود و فرض می‌شودکه آن FS باشد.
۲. یک عدد تصادفی بین 0 و FS تولید می‌شود.
۳. مقادیر برازندگی تمامی کروموزوم‌ها یک به یک تا زمانی‌که حاصل بزرگ‌تر یا مساوی عدد تصادفی تولید شده باشد جمع می‌شوند.
۴. آخرین کروموزوم انتخاب می‌شود.

این فرآیند تا زمانی‌که به تعداد کافی کروموزوم انتخاب شود تکرار می‌شود.

در پیاده‌سازی چرخ رولت با این روش، افراد روی چرخ رولت قرار داده می‌شوند و بر طبق مقادیر برازندگی بخشی از چرخ را اشغال می‌کنند. سپس چرخ چرخانده می‌شود و بعد از توقف، کروموزوم نشان داده شده به وسیله شاخص انتخاب می‌شود. فرآیند انتخاب چرخ رولت را به تعداد لازم می‌چرخاند و در هر بار کروموزومی را که شاخص چرخ رولت پس از متوقف شدن چرخ به آن اشاره می‌کند انتخاب می‌کند. در شکل ۲-۲۷ مثالی از چرخ رولت نمایش داده شده است [۱۴].



شکل۲-۲۷: روش انتخاب چرخ رولت [۱۲].

۲-۹-۲ روش دوره‌ای[۱]

این روش برای کاهش احتمال هم‌گرایی زودرس پیشنهاد شده است. در این روش دو کپی از جمعیت نگه‌داری می‌شود. در هر نسل کروموزوم‌های مجاور در یک کپی دو به دو با هم مقایسه می‌شوند و کروموزومی را که مقدار برازندگی آن بیشتر است انتخاب می‌شود. سپس همین فرآیند در کپی دوم

[۱] tournament selection

جمعیت نیز انجام می‌شود تا دو کپی جدید که تعداد کروموزوم‌های هر یک از آن‌ها نصف دو کپی اولیه می‌باشد به دست آید. فرآیند فوق تا زمانی‌که تعداد کروموزوم‌های هر یک از کپی‌ها برابر یک شود تکرار می‌شود و در نهایت از دو کپی، کروموزومی انتخاب می‌شود که مقدار برازندگی آن بیشتر است. مزیت این روش این است که اگر کروموزوم‌های متوسط با کروموزوم‌های نامناسب مقایسه شوند آن‌ها نیز شانس کمتری برای انتخاب دارند.

روش دوره‌ای k مسیر[۱]: در این روش به طور تصادفی k کروموزوم انتخاب می‌شوند و سپس از میان این k کروموزوم، دو شخصی را که دارای بالاترین مقدار برازندگی هستند انتخاب می‌شوند. این روش همیشه به بهترین راه حل‌ها توجه نمی‌کند.

۲-۹-۳ روش رتبه‌بندی[۲]

زمانی‌که مقادیر برازندگی اختلاف زیادی دارند، روش‌های انتخاب قبلی دارای مشکلاتی خواهد بود. برای مثال اگر شایستگی بهترین کروموزوم در روش‌های قبلی ۹۰٪ باشد در این صورت کروموزوم‌های دیگر شانس خیلی کمتری برای انتخاب شدن خواهند داشت. روش رتبه‌بندی در ابتدا جمعیت ژنتیکی را دسته بندی می‌کند و سپس هر کروموزوم، یک مقدار برازندگی را از بین دسته بندی به خود می‌گیرد. در بدترین حالت، رتبه برابر ۱ خواهد بود و بعد ۲ و ... . بهترین رتبه برابر n خواهد بود که n تعداد کروموزوم‌ها در جمعیت است. در این صورت تمامی کروموزوم‌ها یک فرصت و شانس برای انتخاب شدن خواهند داشت. انتخاب با جایگزینی می‌باشد که اجازه می‌دهد اشخاص بیش از یک بار انتخاب شوند. عیب این روش این است که ممکن است به هم‌گرایی کند و آهسته منجر شود چون بهترین کروموزوم‌ها اختلاف زیادی با هم‌دیگر نخواهند داشت.

۲-۱۰ عمل‌گر ترمیم[۳]

ممکن است پس از انجام عمل‌گرهای جهش و ترکیب، راه‌حل‌های نامعتبر ایجاد شوند. برای رفع این مشکل از عمل‌گر ترمیم استفاده می‌شود تا راه حل ایجاد شده معتبر شود [۱۴].

---

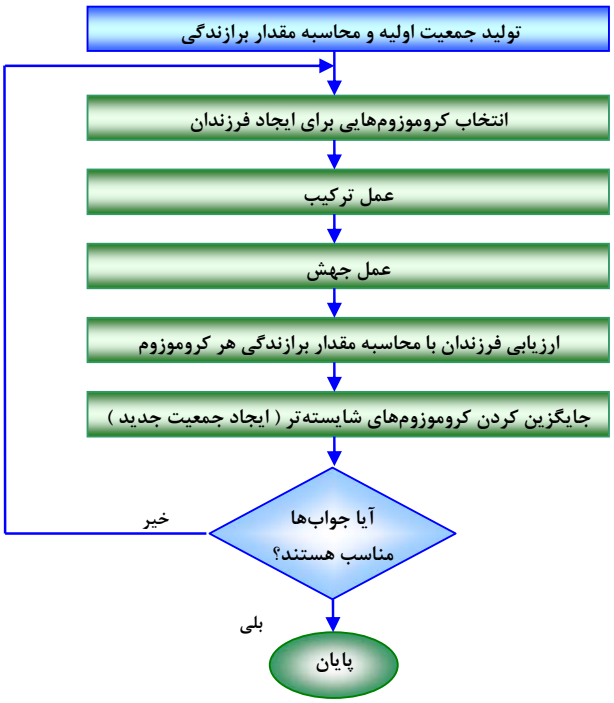
۱ k-way tournament
۲ ranking selection
۳ repair

۲-۱۱ نخبه کشی[۱]

برای این که راه‌حل‌های بهتر از بین نروند، کروموزومی که در نسل جدید دارای پایین‌ترین مقدار برازندگی باشد حذف شده و کروموزومی را که در نسل قبلی دارای بالاترین مقدار برازندگی است، جایگزین آن می‌شود [۱۴].

۲-۱۲ مراحل اجرای الگوریتم ژنتیک

پس از بیان مفاهیم اولیه، مراحل مختلف در استفاده از الگوریتم ژنتیک مورد بررسی قرار می‌گیرد. ابتدا با توجه به صورت مساله، متغیرهایی که باید تعیین شوند، مشخص می‌شوند. سپس این متغیرها به نحو مناسبی کدگذاری شده و به شکل کروموزوم نمایش داده می‌شوند. بر اساس تابع هدف، یک تابع برازندگی برای کروموزوم‌ها تعریف می‌گردد و یک جمعیت اولیه دلخواه نیز به طور تصادفی انتخاب می‌شود. به دنبال آن، میزان تابع برازندگی برای هر کروموزوم جمعیت اولیه محاسبه می‌شود. سپس مراحلی که در شکل ۲-۲۸ نمایش داده شده است به ترتیب زیر انجام می‌گیرد. آن چه که تاکنون به عنوان ساختار کلی الگوریتم ژنتیک ارایه شده است، به وسیله گرفنستت[۲] و بیکر[۳] ارایه گردیده است و خلاصه آن به صورت الگوریتم نمایش داده شده در شکل ۲-۲۹ می‌باشد.



شکل ۲-۲۸: مراحل اجرای الگوریتم ژنتیک [۲].

---

۱ elitism
۲ Gerfenstet
۳ Baker

```
Genetic Algorithm
1. Begin
2.   t=0;
3.   initialize P(t);
4.   evaluate P(t);
5.   while (recombination condition) {
6.         recombination(t)  to yield  C(t);
7.         evaluate C(t);
8.         select  P(t+1)   from   P(t) , C(t);
9.   }//end of while
10. End.
```

شکل ۲-۲۹: الگوریتم ژنتیک ارایه شده به وسیله گرفنستت و بیکر

مرحله ۱: در این مرحله تعداد مناسبی از زوج کروموزوم‌ها بر اساس میزان برازندگی آن‌ها انتخاب می‌شوند تا در مراحل بعدی مورد استفاده قرار بگیرند. کروموزوم‌هایی که دارای مقدار برازندگی بالایی هستند، ممکن است چندین بار در مراحل تولید انتخاب شوند، در حالی که کروموزوم‌هایی که مقدار برازندگی آن‌ها کم می‌باشد، ممکن است هیچ‌گاه انتخاب نگردند. ساده‌ترین روش برای اجرای این مرحله، استفاده از مدل چرخ رولت است که در بخش ۲-۹-۱ معرفی گردید.

مرحله ۲: در این مرحله عمل‌گر ترکیب با احتمال P_c بر روی کروموزوم‌های والد عمل کرده و با ترکیب آن‌ها، کروموزوم‌های جدیدی (فرزندان) را تولید می‌کند. در عمل ترکیب، اطلاعات جدید به طور معمول فقط بر اساس اطلاعات موجود در کروموزوم‌های فعلی (کروموزوم‌های حاضر در جمعیت والدین) استخراج می‌گردد. چنان‌چه اطلاعات خاصی به دلایلی مثل:

- ❖ محدودیت در ذخیره‌سازی اطلاعات (محدودیت در تعداد اعضای جمعیت)
- ❖ از دست رفتن اطلاعات در مرحله انتخاب (به دلیل آن که این اطلاعات در کروموزوم‌هایی با برازندگی کم قرار دارد.)

از بین برود آنگاه عمل‌گر ترکیب قادر نخواهد بود تا ساختارهای جدیدی را که حاوی اطلاعات از دست رفته باشند، به وجود آورد.

مرحله ۳: در این مرحله عمل جهش با احتمال P_m بر روی کروموزوم‌های حاصل از عمل ترکیب انجام شده و با تغییر بیت‌های این کروموزوم‌ها، راهی را برای ورود اطلاعات جدید به وجود می‌آورد.

مرحله ۴: در این مرحله به منظور ارزیابی فرزندان، مقدار برازندگی کروموزوم‌های جدید محاسبه می‌گردد.

مرحله ۵: در این مرحله جمعیت جدید برای ورود به مرحله بعد الگوریتم، انتخاب می‌گردد. این کار با مقایسه مقدار برازندگی کروموزوم‌ها انجام می‌شود. روش‌های مختلفی برای انتخاب جمعیت جدید وجود دارد که به طور مثال می‌توان دو روش زیر را نام برد:

- ❖ تمام اعضای جمعیت جدید از میان کروموزوم‌های فرزندان انتخاب می‌شوند.
- ❖ تعدادی از افراد جمعیت مرحله بعد، همان افراد جمعیت مرحله قبل هستند و بقیه از میان فرزندان جدید انتخاب می‌گردد. البته در این مورد، شایسته‌ترین کروموزوم‌ها انتخاب می‌شوند.

تحقیقات نشان داده است که حذف همه کروموزوم‌های جمعیت مرحله قبل و انتخاب جمعیت جدید از میان فرزندان، ممکن است بسیاری از جواب‌های مناسب را که در میان جمعیت مرحله قبل وجود دارد، حذف نماید. بنابراین پیشنهاد می‌شود که چنان‌چه از روش اول برای انتخاب جمعیت جدید استفاده می‌شود، در هر مرحله، بهترین جواب‌ها دخیره شوند و چنان‌چه در مراحل بعدی، جواب‌های بهتری به دست آمده، آن‌ها را جایگزین جواب‌های ذخیره شده کنند. این کار مانع از دست رفتن اطلاعات در مرحله انتخاب می‌شود.

مرحله ۶: در این مرحله همه افراد جمعیت جدید مورد ارزیابی قرار می‌گیرند. چنان‌چه شرایط خاتمه الگوریتم فراهم باشد، الگوریتم پایان می‌پذیرد و در غیر این صورت جمعیت موجود به عنوان جمعیت اولیه برای مرحله بعد مورد استفاده قرار می‌گیرد. شرایط خاتمه الگوریتم ژنتیک می‌تواند به وسیله مساله مشخص شود و یا شرایطی مانند زمان اجرای الگوریتم، تعداد محدودی تولید در انجام الگوریتم و یا تغییر نکردن بهترین جواب برای تعداد مشخصی از مراحل تولید باشد.

۲-۱۳ محدودیت‌های الگوریتم ژنتیک

یک مشکل در الگوریتم ژنتیک چگونگی نوشتن تابع برازندگی است، که منجر به بهترین راه حل برای مساله شود. اگر تابع ارزیابی به خوبی و قوی انتخاب نشود، ممکن است باعث شود که راه حلی برای مساله پیدا نشود. به علاوه برای انتخاب تابع ارزیابی مناسب، پارامترهای دیگری مثل اندازه جمعیت، نرخ جهش، نرخ ترکیب، قدرت و نوع انتخاب هم باید مورد توجه قرار گیرند.

مشکل دیگر این است که اگر یک کروموزوم فاصله‌اش با سایر کروموزوم‌های نسل خودش زیاد باشد (خیلی بهتر از بقیه باشد) و خیلی زود دیده شود (ایجاد شود) ممکن است محدودیت ایجاد کند و راه حل را به سوی جواب بهینه محلی سوق دهد. این اتفاق به طور معمول در جمعیت‌های کمتر اتفاق می‌افتد.

۲-۱۴ هم‌گرایی در الگوریتم ژنتیک

سوال مهمی که می‌تواند مطرح شود این است که آیا الگوریتم ژنتیک همواره به سمت بهینه مطلق هم‌گرا می‌شود؟ تحقیقات رونکلف در سال ۱۹۹۴، هم‌گرایی الگوریتم ژنتیک را تحت شرایط خاص به اثبات می‌رساند. در این تحقیقات، نشان داده شده است که هم‌گرایی به سمت بهینه مطلق یک

خاصیت ذاتی الگوریتم ژنتیک نمی‌باشد ولی با رعایت شرایط خاصی امکان‌پذیر است. تحلیل‌های ریاضی انجام گرفته شده در قالب چندین قضیه و با استفاده از مدل زنجیره مارکوف، در این تحقیق نشان داده است که چنان‌چه در هر مرحله تولید از الگوریتم ژنتیک، بهترین جواب‌ها نگاه داشته و با احتمال یک، به مرحله بعد وارد شوند، الگوریتم به سمت بهینه مطلق هم‌گرا می‌شود.

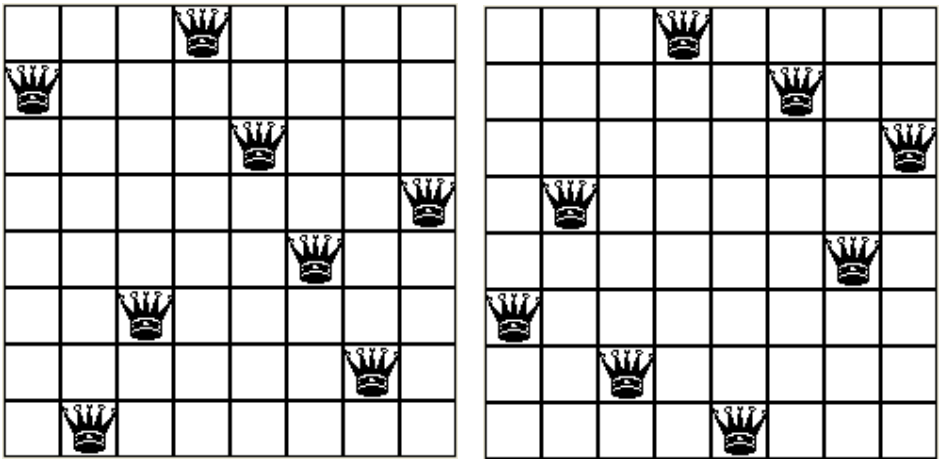
عوامل متعددی در سرعت هم‌گرایی الگوریتم ژنتیک موثر هستند. جمعیت اولیه، مقادیر احتمال جهش و ترکیب، چگونگی انجام عمل ترکیب و جهش، تابع برازندگی و چگونگی انتخاب جمعیت بعدی از جمله این عوامل می‌باشند. چگونگی تاثیر این عوامل در سرعت هم‌گرایی الگوریتم ژنتیک، به مساله مورد نظر بستگی دارد و از طریق آزمایش به دست می‌آید.

۲-۱۵ چند مثال برای الگوریتم ژنتیک

در این بخش سه مساله متفاوت معرفی شده و سپس با استفاده از الگوریتم ژنتیک سعی می‌شود تا جواب قابل قبولی حاصل شود.

۲-۱۵-۱ مساله اول (مساله هشت وزیر)

در این بخش یک الگوریتم ژنتیک برای مساله هشت وزیر معرفی می‌شود. در مساله هشت وزیر هدف قرار دادن هشت مهره وزیر در خانه‌های شطرنج است، به طوری‌که هیچ کدام از این مهره‌ها با هم‌دیگر برخورد نداشته باشند. برای این مساله ۹۲ جواب مختلف وجود دارد که دو نمونه از آن‌ها در شکل۲-۳۰ نمایش داده شده است. در الگوریتم ژنتیک معرفی شده در این بخش هدف پیدا کردن اولین راه حل برای مساله می‌باشد، بنابراین الگوریتم زمانی خاتمه پیدا می‌کند که به اولین جواب مورد نظر برسد.



(ب) (الف)

شکل ۲-۳۰: دو راه حل مختلف برای مساله هشت وزیر

۲-۱۵-۱-۱ کدگذاری مساله

برای این مساله هر کروموزوم از هشت ژن تشکیل شده است، به طوری که ژن اول محل مهره اول و ژن دوم محل مهره دوم و در نهایت ژن هشتم محل مهره هشتم در هر سطر را مشخص می‌کند. در واقع هر کروموزوم نمایش دهنده یک جواب برای مساله است که ممکن است قابل قبول و یا غیر قابل قبول باشد. برای مثال کروموزوم مربوط به راه حل نمایش داده شده در شکل ۲-۳۰ به صورت شکل ۲-۳۱ می‌باشد.

(ب) (الف)

شکل ۲-۳۱: کروموزوم‌های مربوط به راه‌حل نمایش داده شده در شکل ۲-۳۰

۲-۱۵-۱-۲ مراحل الگوریتم

در شکل ۲-۳۲ الگوریتم ژنتیک برای مساله هشت وزیر به طور کامل آورده شده است. همان‌طور که مشاهده می‌شود الگوریتم زمانی خاتمه پیدا می‌کند که به یکی از راه حل‌های مورد نظر برسد. در غیر این صورت اجرای الگوریتم ادامه پیدا می‌کند. لازم به ذکر است که قبل از اجرای الگوریتم ژنتیک، می‌باید اندازه جمعیت اولیه و نرخ عمل ترکیب و نرخ عمل جهش مشخص شده باشد.

۱. به اندازه جمعیت اولیه کروموزوم تصادفی تولید کن.
۲. تعداد برخوردهای هر کروموزوم را محاسبه کن و آن را در S قرار بده.
۳. مقدار برازندگی هر کدام از کروموزوم‌ها را محاسبه کن (مقدار برازندگی هر کروموزوم عبارت است از: $\frac{1}{1+S}$).
۴. درصورتی که مقدار برازندگی یکی از کروموزوم‌ها برابر ۱ باشد، به مرحله ۹ برو.
۵. انجام عمل انتخاب.
۶. به اندازه نرخ ترکیب عمل ترکیب را انجام بده.
۷. به اندازه نرخ جهش عمل جهش را انجام بده.
۸. جمعیت جدید را بر روی جمعیت قبلی کپی کن. برو به مرحله ۲.
۹. پایان

شکل ۲-۳۲: الگوریتم ژنتیک برای مساله هشت وزیر

۲-۱۵-۱-۲-۱ تولید جمعیت اولیه

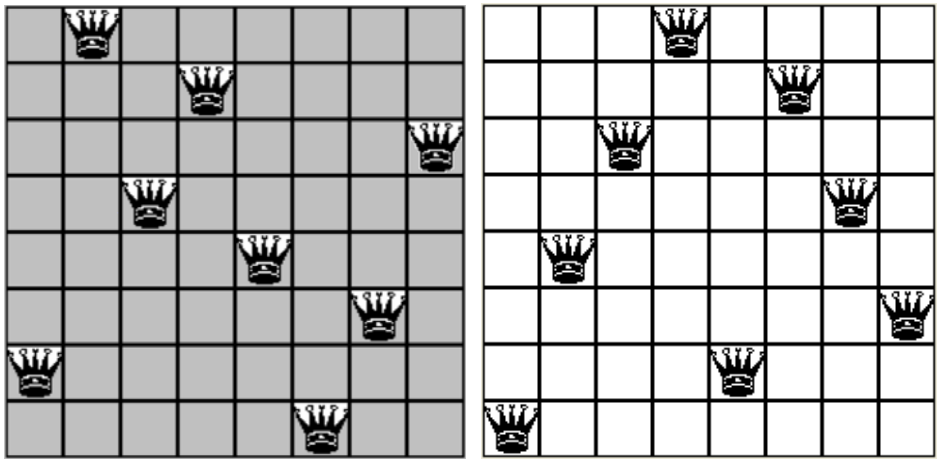
در ابتدا تعدادی کروموزوم تصادفی به اندازه جمعیت اولیه تولید می‌شود، که هر کدام از این کروموزوم‌ها نمایش دهنده یک راه‌حل می‌باشند که می‌توانند قابل قبول و یا غیر قابل قبول باشند.

۲-۱۵-۱-۲-۲ تابع برازندگی

مقدار برازندگی هر کروموزوم عبارت است از $\frac{1}{1+S}$، که در آن S عبارت است از تعداد کل برخوردهای کروموزوم‌ها. در صورتی که کروموزومی دارای هیچ برخوردی نباشد در این صورت مقدار برازندگی آن برابر ۱ می‌شود. علت جمع شدن عدد 1 با S در مخرج این است که ممکن است در مساله هیچ برخوردی اتفاق نیفتد و مقدار S برابر صفر باشد، بنابراین اگر عدد 1 با S جمع نشود خطای تقسیم بر صفر اتفاق می‌افتد.

۲-۱۵-۱-۲-۳ عمل‌گر ترکیب

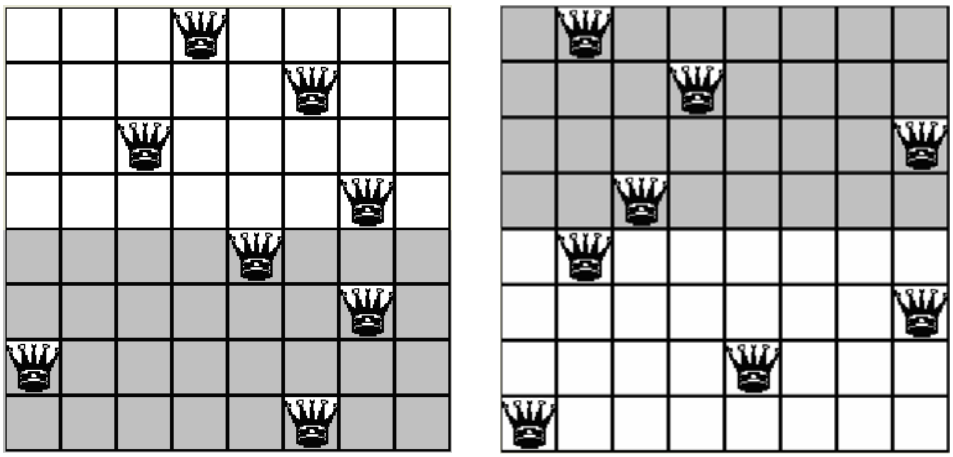
در این مساله از ترکیب تک نقطه‌ای برای انجام عمل ترکیب استفاده شده است. در زیر نحوه ترکیب تک نقطه‌ای با استفاده از یک مثال بر روی مساله هشت وزیر توضیح داده شده است. هدف ترکیب دو کروموزوم نمایش داده شده در شکل ۲-۳۳ است. کروموزوم‌های حاصل شده از ترکیب آن‌ها در شکل ۲-۳۵ نمایش داده شده است.



(ب) (الف)

شکل۲-۳۳: نمایش دو کروموزوم قبل از عمل ترکیب

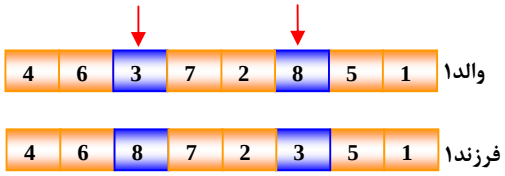
شکل۲-۳۴: ترکیب تک نقطه‌ای دو کروموزوم شکل ۲-۳۳



شکل۲-۳۵: نمایش دو کروموزوم بعد از عمل ترکیب

۲-۱۵-۱-۲-۴ عمل‌گر جهش

پس از انجام عمل ترکیب بر روی کروموزوم‌ها، نوبت به انجام عمل جهش می‌رسد. برای این مساله از روش تعویض ژن‌ها جهت عمل‌گر جهش استفاده شده است. شکل ۲-۳۶ نحوه عمل جهش به روش تعویض ژن‌ها را نمایش می‌دهد.



شکل۲-۳۶: نحوه انجام عمل جهش

۲-۱۵-۱-۲-۵ فرآیند انتخاب

برای این مساله از روش چرخ رولت جهت انتخاب کروموزوم‌ها و انتقال آن‌ها به نسل بعدی استفاده شده است.

۲-۱۵-۲ مساله دوم (بیشینه سازی مقدار تابع)

در این مساله هدف بیشینه کردن مقدار تابع زیر می‌باشد.

$$f\ (x\ ,\ y\ ) = \sqrt{(x^{\ 2}\ +\ y^{\ 2})} \qquad (۱-۲)$$

در این تابع، x یک عدد صحیح در فاصله [0, 31] و y یک عدد صحیح در فاصله [0, 7] می‌باشد. برای استفاده از الگوریتم ژنتیک، جهت بیشینه کردن مقدار $f\ (x\ ,\ y\ )$، ابتدا لازم است متغیرهای x و y با دنباله‌ای مناسب از اعداد دودویی نمایش داده شوند. به طور کلی تعداد بیت در نظر گرفته شده برای هر کروموزوم، به دقت لازم برای جواب بستگی دارد. در این مثال هر عدد صحیح بین 0 و 31 را می‌توان به وسیله پنج بیت و هر عدد صحیح بین 0 و 7 را با سه بیت نمایش داد. بنابراین هر نقطه در فضای دو بعدی $0 < y < 7$ و $0 < x < 31$ به صورت شکل ۲-۳۷ کدگذاری می‌شود.

$(x\ ,\ y\ )$: (□□□□□ & □□□) → □□□□□□□□

شکل ۲-۳۷: نمایش کروموزوم معادل زوج (x, y)

چون هدف بیشینه کردن تابع $f\ (x\ ,\ y\ )$ است، در این صورت تابع برازندگی باید یک تابع صعودی از $f\ (x\ ,\ y\ )$ باشد. به عنوان نمونه می‌توان توابع زیر را برای تابع برازندگی پیشنهاد کرد:

$$Fitness\ = \frac{f\ (x\ ,\ y\ )}{\max f\ (x\ ,\ y\ )} \qquad (۲-۲)$$

$$Fitness\ = \exp[-(\frac{(1/f\ (x\ ,\ y\ ) - 1)}{\max f\ (x\ ,\ y\ )})] \qquad (۳-۲)$$

در این مثال، تابع برازندگی به صورت زیر در نظر گرفته شده است:

$$Fitness\ = \frac{\sqrt{x^{\ 2}\ +\ y^{\ 2}}}{\sqrt{31^2\ +\ 7^{\ 2}}} \qquad (۴-۲)$$

برای انجام عمل ترکیب و جهش، $p_c = 0.7$ و $p_m = 0.2$ فرض شده است و محل اتصال دو متغیر x و y به عنوان نقطه ترکیب، انتخاب شده است. جمعیت اولیه که به صورت تصادفی انتخاب شده است، در جدول شکل ۲-۳۸ نمایش داده شده است.

شماره	(x, y)	کروموزوم معادل	مقدار برازندگی
1	(1, 3)	00001011	0.0995
2	(5, 1)	00101001	0.1604
3	(0, 2)	00000010	0.0629

شکل ۲-۳۸: نمایش جمعیت اولیه

برازندگی کروموزوم‌های جدید	نتایج حاصل از عمل ترکیب		کروموزوم‌های شرکت کننده در عمل ترکیب
	کروموزوم معادل	(x, y)	
0.0445	00001001	(1, 1)	{1, 2}
0.1835	00101011	(5, 3)	
0.0703	00001010	(1, 2)	{1, 3}
0.0944	00000011	(0, 3)	
0.1649	00101010	(2, 5)	{2,3}
0.0315	00000001	(0, 1)	

شکل ۲-۳۹: نتایج حاصل شده از عمل ترکیب

برازندگی	(x, y) حاصل	اعداد تصادفی								(x, y) اولیه
0.2225	(5, 5)	0.9501	0.0185	0.2311	0.5028	0.6721	0.7095	0.4860	0.4289	(1, 1)
0.1835	(5, 3)	0.8318	0.2311	0.8214	0.4057	0.8132	0.1988	0.8462	0.3759	(5, 3)
0.6011	(19 ,2)	0.2026	0.5028	0.6068	0.4447	0.9355	0.0099	0.0153	0.5252	(1, 2)
0.1573	(4, 3)	0.4451	0.6721	0.7095	0.4860	0.6154	0.9169	0.1389	0.7468	(0, 3)
0.3518	(11, 2)	0.1987	0.9318	0.8381	0.4289	0.8913	0.7919	0.4103	0.2028	(2, 5)
0.2688	(8, 3)	0.0579	0.6038	0.4660	0.0196	0.3046	0.7621	0.9218	0.8936	(0, 1)

شکل ۲-۴۰: نتایج حاصل شده از عمل جهش

حال از بین کروموزوم‌های جدید به دست آمده از عمل ترکیب و جهش و کروموزوم‌های اولیه، کروموزوم با بیشترین مقدار برازندگی به عنوان جمعیت جدید برای مرحله بعدی الگوریتم انتخاب می‌گردد (شکل ۲-۴۱).

شماره	(x, y)	کروموزوم معادل	مقدار برازندگی
1	(19, 2)	10011010	0.6011
2	(8, 3)	01000011	0.2888
3	(11, 2)	01011010	0.3518

شکل ۲-۴۱: جمعیت جدید حاصل شده

با مقایسه جدول نمایش داده شده در شکل ۲-۳۸ و شکل ۲-۴۱ مشاهده می‌شود که کروموزوم‌های حاصل از یک بار انجام مراحل الگوریتم، دارای مقدار برازندگی بهتری نسبت به کروموزوم‌های اولیه هستند. کروموزوم‌های جدید، جمعیت اولیه مرحله بعد می‌باشند. عمل ترکیب و جهش بر روی آن‌ها انجام شده و بهترین نتایج، انتخاب می‌گردد. آن قدر این عملیات ادامه می‌یابد تا در نهایت، یک مقدار بیشینه برای تابع مورد نظر به دست آید. در این حالت، الگوریتم پایان می‌پذیرد.

۲-۱۵-۳ مساله سوم (مربع ۳×۳)

هدف از این مساله جایگزینی اعداد بین ۱ تا ۱۵ در خانه‌های یک مربع ۳×۳ است، به طوری که مجموع تمام سطرها و ستون‌های مربع برابر با عدد ۲۴ باشد (شکل ۲-۴۲).

8	10	6	= 24
12	5	7	= 24
4	9	11	= 24
||	||	||	
24	24	24	

شکل ۲-۴۲: ساختار مربع ۳×۳

این مساله تا حدودی پیچیده است. ممکن است یک انسان بتواند آن را در مدت زمانی مشخص حل کند ولی هیچ گاه یک کامپیوتر نخواهد توانست آن را در مدت زمان کوتاهی با استفاده از اعداد تصادفی حل کند. ولی الگوریتم ژنتیک می‌تواند یک جواب قابل قبولی برای این مساله پیدا کند. در زیر یک الگوریتم ژنتیک برای حل این مساله معرفی خواهد شد.

۲-۱۵-۳-۱ تولید جمعیت اولیه

اولین گام ایجاد کردن یک جمعیت اولیه برای شروع کار است، که شامل تعدادی کروموزوم تصادفی است. این کروموزوم‌ها به صورت دودویی نشان داده می‌شوند. در ابتدا یک سری اعداد به صورت تصادفی تولید می‌شوند. هر کروموزوم شامل اطلاعاتی برای هر 9 خانه مربع است. چون این اعداد مقادیر بین 0 تا 15 را دارند، در این صورت می‌توان آن‌ها را با 4 بیت یا ژن نمایش داد. پس هر کروموزوم شامل $9 \times 4 = 36$ ژن می‌باشد. در شکل ۲-۴۳ مثالی از یک کروموزوم برای این مساله نمایش داده شده است.

ژن‌های کروموزوم‌ها به صورت اعداد در مبنای ۲

0110	**1100**	**1111**	**1011**	**0100**	**1010**	**0111**	**0101**	**1110**

ژن‌های کروموزوم‌ها به صورت اعداد در مبنای ۱۰

6	**12**	**15**	**11**	**4**	**10**	**7**	**5**	**14**

شکل ۲-۴۳: نمایش یک کروموزوم برای مساله مربع ۳×۳

۲-۱۵-۳-۲ تابع برازندگی

حال باید مقدار برازندگی برای تمام کروموزوم‌ها با استفاده از تابع برازندگی محاسبه شود. انتخاب تابع برازندگی یکی از کارهای سخت و حساس مساله می‌باشد. برای این مساله تابع برازندگی به صورت زیر می‌باشد:

$$Fitness = \frac{1}{1 + \sum_{i=0}^{2} \left|(sum(row_i) - 24\right| + \sum_{i=0}^{2} \left|(sum(column_i) - 24\right|} \qquad (۵-۲)$$

در صورتی که این مساله به مربع $n \times n$ تغییر کند در آن صورت تابع برازندگی آن به صورت رابطه زیر تغییر پیدا می‌کند:

$$Fitness = \frac{1}{1 + \sum_{i=0}^{n-1} \left|(sum(row_i) - 24\right| + \sum_{i=0}^{n-1} \left|(sum(column_i) - 24\right|} \qquad (۶-۲)$$

6	12	15	= 33
11	4	10	= 25
7	5	14	= 26
||	||	||	
24	21	39	

شکل ۲-۴۴: مثال برای مربع ۳×۳

برای مثال اگر اعداد موجود در خانه‌های مربع به صورت شکل ۲-۴۴ باشد در آن صورت مقدار تابع برازندگی آن طبق رابطه (۲-۵) به صورت زیر می‌شود.

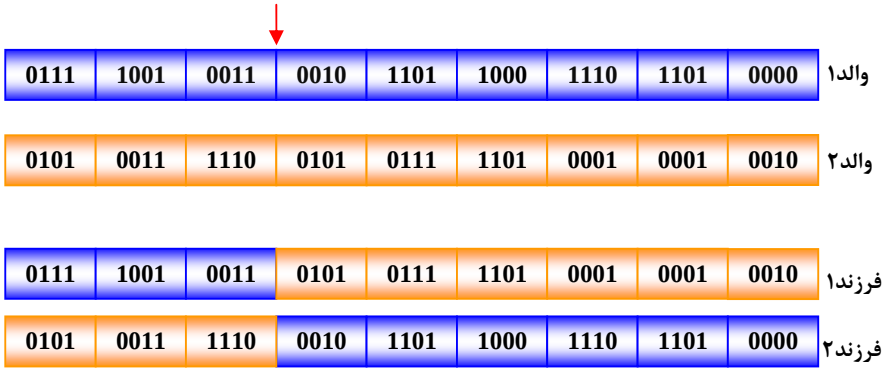
$$Fitness = \frac{1}{1 + |33 - 24| + |25 - 24| + |26 - 24| + |24 - 24| + |21 - 24| + |39 - 24|}$$

$$= \frac{1}{1 + 9 + 1 + 2 + 0 + 3 + 15} = \frac{1}{31} \approx 0.032$$

بنابراین مقدار برازندگی برای این کروموزوم تقریباً برابر 0.032 است. در صورتی که مقدار حاصل شده به وسیله تابع برازندگی برابر ۱ باشد، در این صورت آن کروموزوم به جواب مورد نظر دستیابی پیدا کرده است.

۲-۱۵-۳-۳ عمل‌گر ترکیب

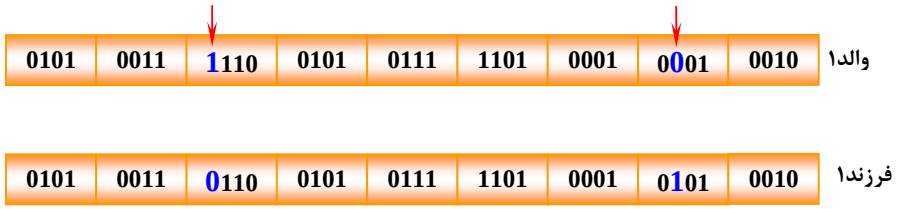
در این مرحله به اندازه نرخ ترکیب دو جفت کروموزوم انتخاب شده و سپس با هم‌دیگر ترکیب می‌شوند. در این مساله از ترکیب تک نقطه‌ای برای انجام عمل ترکیب استفاده شده است. در شکل ۲-۴۵ نحوه ترکیب تک نقطه‌ای با استفاده از یک مثال برای مساله مورد نظر توضیح داده شده است.



شکل ۲-۴۵: اعمال عمل‌گر ترکیب بر روی دو کروموزوم

۲-۱۵-۳-۴ عمل‌گر جهش

پس از انجام عمل ترکیب بر روی کروموزوم‌ها، نوبت به انجام عمل جهش می‌رسد. برای این مساله از روش تعویض ژن‌ها جهت عمل‌گر جهش استفاده شده است. شکل ۲-۴۶ نحوه عمل جهش به روش تعویض ژن‌ها را نمایش می‌دهد.



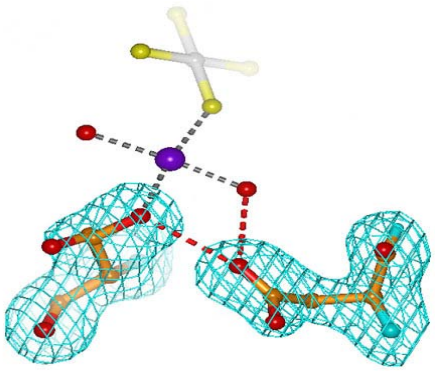
شکل ۲-۴۶: اعمال عمل‌گر جهش بر روی دو کروموزوم

۲-۱۵-۳-۵ فرآیند انتخاب

برای این مساله از روش چرخ رولت جهت انتخاب کروموزوم‌ها و انتقال آن‌ها به نسل بعدی استفاده شده است.

۲-۱۶ خلاصه فصل

در این فصل مفهوم پردازش تکاملی و الگوریتم ژنتیک مطرح شدند. هم‌چنین روش‌های مختلف کدگذاری برای الگوریتم ژنتیک شرح داده شدند. تعداد مختلفی از عمل‌گرهای ترکیب و جهش و فرآیند انتخاب که از مهم‌ترین بخش‌های الگوریتم ژنتیک محسوب می‌شوند معرفی شدند. در پایان سه مثال مختلف به همراه راه حل آن به وسیله الگوریتم ژنتیک شرح داده شدند. در ادامه در فصل آتی الگوریتم شبیه‌ساز سرد کردن فلزات معرفی خواهد شد.



فصل سوم

الگوریتم شبیه‌ساز سرد کردن فلزات

۳-۱ مقدمه

الگوریتم شبیه‌ساز سرد کردن فلزات (SA[۱]) یا به عبارتی دیگر الگوریتم گدازش/سرمایش در اوایل دهه ۱۹۸۰ به وسیله *کیرکپاتریک*[۲] و همکارانش ارایه شد [۱۶، ۱۸، ۱۹]. این روش، فرآیند شبیه‌ساز سرد کردن فلزات، مواد را شبیه‌سازی می‌کند. طی فرآیند شبیه‌ساز سرد کردن فلزات، یک ماده تا دمایی بیشتر از دمای ذوبش گرم می‌شود و سپس به تدریج، دمای آن پایین آورده می‌شود. نحوه‌ی کاهش دما بسیار کند و در حدی است که ماده در تعادل ترمودینامیکی است. به عبارت دیگر، دمای جسم آن قدر ثابت می‌ماند که بهترین ساختار بلوری با کم‌ترین انرژی در آن دما تشکیل شود. اجسامی که ساختار بلوری‌شان در انرژی‌های بالاتری شکل گرفته باشد، شکننده‌تر نیز هستند. اما بر عکس، اگر ساختار بلوری جسمی، در انرژی‌های کم‌تر تشکیل شده باشد، از مقاومت فیزیکی بسیار بیشتری برخوردار خواهد بود [۱].

در الگوریتم شبیه‌ساز سرد کردن فلزات، جواب‌های پیشنهادی برای مساله، در دمای بالاتر قرار دارند و اغلب جواب‌های مناسبی نیستند. این نامناسب بودن را می‌توان به شکنندگی تشبیه نمود. سپس متغیری که نقش دما را بر عهده دارد به مرور زمان کاهش داده می‌شود تا به این ترتیب جواب‌های بهتری در دماهای پایین تشکیل شوند. الگوریتم، جواب‌های بعدی را با استفاده از جواب‌های فعلی به دست می‌آورد. در این الگوریتم، گرم کردن با اعمال تغییرات تصادفی بیشتر بر روی متغیرها مترادف

---

[۱] Simulated Annealing

[۲] Kirkpatrick

است. در دماهای بالاتر، متغیرها دارای تغییرات زیادتری هستند و هر چه دما پایین‌تر می‌آید، دامنه تغییرات تصادفی متغیرها نیز، کم‌تر و کم‌تر می‌شود. همواره تغییراتی که منجر به بهتر شدن نتیجه شوند، پذیرفته می‌شوند. اما تغییراتی که منجر به بدتر شدن نتیجه شوند، با احتمالی پذیرفته می‌شوند. این احتمال نیز با کم‌تر شدن دما، کم‌تر می‌شود. یک ازدیاد در مقدار تابع هزینه، به مقدار Δ، فقط هنگامی پذیرفته می‌شود که شرط

$$e^{-\frac{\Delta}{T}} > r \qquad (۱-۳)$$

برقرار باشد، که در آن T دما و r یک متغیر تصادفی با توزیع یکنواخت در بازه‌ی [0, 1] است.

به طور معمول در ابتدای الگوریتم $T = 1$ در نظر گرفته می‌شود. الگوهای فراوانی برای کاهش دما وجود دارند که از میان آن‌ها می‌توان به موارد زیر اشاره کرد. در هر یک از موارد زیر، فرض بر این است که دما در زمان n برابر با T_n و N آخرین مرحله از اجرای الگوریتم باشد [۱۶، ۱۸].

کاهش خطی: در این روش دماهای مراحل مختلف، اعضای یک دنباله‌ی حسابی هستند.

$$T_n = T_0 - \frac{n(T_0 - T_N)}{N} \qquad (۲-۳)$$

کاهش هندسی: در این روش دماهای مراحل مختلف، یک دنباله‌ی هندسی تشکیل می‌دهند.

$$T_n = \lambda^n T_0, \quad 0 \ll \lambda < 1 \qquad (۳-۳)$$

کاهش لگاریتمی: در این روش c کم‌ترین تغییرات لازم برای خارج شدن از کمینه‌های محلی است.

$$T_n = \frac{c}{\log(n+1)} \qquad (۴-۳)$$

نحوه‌ی عملکرد الگوریتم شبیه‌ساز سرد کردن فلزات در شکل (۳-۱) به صورت یک شبه کد آمده است. این الگوریتم برای حل انواع مسایل بهینه‌سازی قابل استفاده است. اولین کاربردی که به وسیله کیرکپاتریک برای الگوریتم شبیه‌ساز سرد کردن فلزات پیشنهاد شد، حل *مساله فروشنده دوره‌گرد* (TSP[۱]) بود [۱، ۲۱]. این مساله یکی از مسایلی است که دارای حل تحلیلی نمی‌باشد و تا به حال هیچ روش مدون ریاضی برای حل آن پیشنهاد نشده است. توضیحات اضافی در مورد این مساله، در ادامه می‌آید.

---

[۱] Traveling Salesman Problem

۳-۲ مساله فروشنده دوره‌گرد

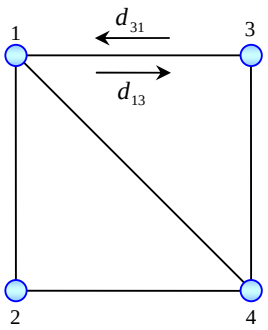
فرض کنید n شهر وجود دارد که فاصله بین آن‌ها معلوم و بین هر شهر مسیری وجود دارد. مسیری را تعیین کنید که فروشنده از یک شهر شروع کند. ضمن اینکه از هر شهر فقط یکبار عبور می‌کند دوباره به شهر اول بازگردد به شرط آنکه هزینه آن مسیر کمترین باشد. در شکل (۳-۲) شکلی از مساله فروشنده دوره‌گرد مشاهده می‌شود. ماتریس D فاصله بین شهرها را مشخص می‌کند. در حالت کلی D می‌تواند غیر متقارن باشد و حتی می‌توان برخی از اعضای این ماتریس را، ∞ (بی‌نهایت) فرض کرد، که به معنی عدم وجود مسیر بین دو شهر است [۵، ۱۷، ۱۹].

شکل ۳-۱: نحوه‌ی عملکرد الگوریتم شبیه‌ساز سرد کردن فلزات

الگوریتم شبیه‌ساز سرد کردن فلزات، در جستجوی یافتن شیاری[۱] است که بتواند در آن حرکت کند و جواب‌های بهتری را پیدا کند و در نهایت به بهترین جواب ممکن برسد. هر چه قدر روند کاهشی دما کندتر باشد، جواب‌هایی که الگوریتم پیدا می‌کند، دقیق‌تر خواهند بود. ثابت شده است که اگر فرآیند کاهش دما به حد کافی کند باشد، جواب بهینه به طور حتم پیدا خواهد شد. به عبارت دیگر هر چه‌قدر

[۱] valley

فرآیند شبیه‌ساز سرد کردن فلزات گسترده‌تر باشد، احتمال آن که الگوریتم به جواب بهینه سراسری برسد، به یک میل می‌کند. اما این نتیجه‌ی نظری و قضیه‌ای که در مورد آن به اثبات رسیده است از نظر عملی کاربرد چندانی ندارد. زیرا زمانی که برای پدیده‌ی شبیه‌ساز سرد کردن فلزات مورد نیاز است اغلب بیشتر از زمان لازم برای جستجوی کامل فضا است. در برخی موارد، بازگشتن به وضعیت‌های به نسبت خوب پیشین، راه حل خوبی برای بهتر شدن نتیجه اجرای الگوریتم است. یکی از نسخه‌های تغییر یافته‌ی این الگوریتم، الگوریتم *شبیه‌ساز سرد کردن فلزات تطبیقی*[۱] است که پارامترهای الگوریتم طی اجرای آن به طور خودکار تغییر می‌کنند. با مقایسه‌ی نحوه‌ی عملکرد الگوریتم ژنتیک و الگوریتم شبیه‌ساز سرد کردن فلزات، می‌توان به این نتیجه رسید که الگوریتم ژنتیک بنابر ماهیت تصادفی‌اش، نقاط جواب را در محیط پراکنده می‌کند و سپس سعی می‌کند که با اعمال تغییرات تصادفی، جواب‌ها را بهتر کند. اما الگوریتم شبیه‌ساز سرد کردن فلزات، همواره روند مشخصی را طی می‌کند و سعی دارد که مسیری را به سمت بهترین جواب ممکن ایجاد کند. در کل این دو الگوریتم از نظر عملکرد قابل مقایسه هستند و هر کدام برای انواع خاصی از مسایل مفید هستند. الگوریتم شبیه‌ساز سرد کردن فلزات در مورد توابعی که دارای نقاط بهینه محلی و چندگانه هستند، بهتر عمل می‌کند. اما برای بهینه‌سازی‌های محلی‌تر، الگوریتم ژنتیک جواب‌های بهتری ارایه می‌دهد [۵، ۱۶، ۱۸].



$$\mathbf{D}=\begin{pmatrix} 0 & d_{12} & d_{13} & d_{14} \\ d_{21} & 0 & \infty & d_{24} \\ d_{31} & \infty & 0 & d_{34} \\ d_{41} & d_{42} & d_{43} & 0 \end{pmatrix}$$

شکل ۳-۲ : یک مساله فروشنده دوره‌گرد با چهار شهر. d_{ij} نشان‌دهنده‌ی فاصله‌ی شهر iاُم از شهر j اُم است. فاصله‌ی هر شهر از خودش صفر است. همیشه d_{ij} با d_{ji} برابر نیست. همان‌طور که در شکل دیده می‌شود، بین شهرهای ۲ و ۳ مسیری وجود ندارد که به صورت فاصله‌ی ∞ (بی‌نهایت) در ماتریس فواصل نشان داده شده است [۱].

یکی دیگر از روش‌های بهینه‌سازی که با روش شبیه‌ساز سرد کردن فلزات ارتباط مستقیم دارد، *ماشین بولتزمان*[۲] می‌باشد که در سال ۱۹۸۳ به وسیله *هینتون*[۳] و *سژنوفسکی*[۴] معرفی گردید [۲۰، ۲۱].

---

[۱] adaptive simulated annealing

[۲] Boltzmann machine

[۳] Hinton

[۴] Sejnowski

ماشین بولتزمان، شبکه‌ای از اجزای مستقل است که به صورت متقارن با وزن‌هایی به هم مربوط می‌باشند. در واقع ماشین بولتزمان، نوعی *شبکه‌ی عصبی بازگشتی*[۱] است که دارای خواص *تصادفی*[۲] نیز می‌باشد. به عبارت بهتر می‌توان ماشین بولتزمان را همتای تصادفی *شبکه‌های هاپفیلد*[۳] در نظر گرفت. شبکه‌ای که ماشین بولتزمان را تشکیل می‌دهد، از اتصال اجزایی تشکیل شده است که همگی دارای دو وضعیت خاموش یا روشن هستند. در ماشین بولتزمان، کمیتی به نام انرژی تعریف می‌شود که بسته به وضعیت اجزای شبکه، تغییر می‌کند. هدف از حل چنین شبکه‌ای، پیدا کردن ترکیبی از وضعیت اجزای شبکه است، به نحوی که شبکه کمترین مقدار انرژی را داشته باشد. وضعیت اجزای شبکه را می‌توان به صورت یک رشته‌ی دودویی در نظر گرفت. به همین دلیل، ماشین بولتزمان برای حل مسایل بهینه‌سازی ترکیبی و گسسته مناسب است [۲۰، ۲۱].

انرژی یک ماشین بولتزمان به این صورت تعریف می‌شود:

$$E \triangleq -\frac{1}{2}\sum_{i,j} w_{ij} s_i s_j + \sum_i \theta_i s_i = -\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i \qquad (۵-۳)$$

که در آن، s_i نشان‌دهنده‌ی وضعیت جزو iاُم است. اگر جزو iاُم خاموش باشد، s_i برابر با صفر خواهد بود و در مقابل، اگر جزو iاُم روشن باشد، s_i برابر با یک در نظر گرفته خواهد شد. w_{ij} وزن ارتباطی میان دو جزو iاُم و jاُم است. ماشین بولتزمان، یک شبکه‌ی متقارن است و همان طور که از رابطه (۳-۵) نیز برمی‌آید، $w_{ij} = w_{ji}$ می‌باشد. هم‌چنین، وزن ارتباطی بین هر جزو و خودش صفر در نظر گرفته می‌شود. یعنی $w_{ii} = 0$ می‌باشد. θ_i نیز *مقدار آستانه‌ی*[۴] مربوط به جزو iاُم می‌باشد. تعریفی که در رابطه (۳-۵) برای انرژی ارایه شده است مشابه با تعریف انرژی در شبکه‌های هاپفیلد می‌باشد. همان طور که گفته شد، تفاوت عمده‌ی ماشین بولتزمان با شبکه‌ی هاپفیلد، تصادفی بودن رفتار اجزای شبکه است.

تغییرات انرژی کل که در اثر تغییر وضعیت جزو iاُم از روشن به خاموش به وجود می‌آید، به نام *شکاف انرژی*[۵] شناخته می‌شود و طبق رابطه (۳-۵) به صورت زیر قابل محاسبه است:

$$\Delta E_i \triangleq E_{s_i=0} - E_{s_i=1} = \sum_j w_{ij} s_j - \theta_i \qquad (۶-۳)$$

همان طور که پیش‌تر گفته شد، ماشین بولتزمان از اجزایی تشکیل شده است که رفتار تصادفی دارند. هر جز با احتمالی روشن یا خاموش است. این احتمال برای جزو iاُم به صورت زیر قابل تعریف است:

---

[۱] recurrent neural network

[۲] stochastic

[۳] hopfield networks

[۴] threshold

[۵] energy gap

$$p_i \triangleq \frac{1}{1+e^{-\frac{\Delta E_i}{T}}} \qquad (۷-۳)$$

که در آن T پارامتری است که نشان دهنده‌ی دمای کلی سامانه می‌باشد. پس از آن که ماشین بولتزمان شروع به کار می‌کند، طی مراحل متوالی، جزئی از اجزای موجود در شبکه انتخاب و با توجه به رابطه (۳-۷) تغییر وضعیت داده می‌شود. پس از آن که در یک دمای معین، این اعمال به حد کافی تکرار شدند، وضعیت احتمالی کل شبکه فقط به انرژی کل شبکه وابسته خواهد بود که به وسیله یک توزیع احتمال بولتزمان قابل توصیف است. در چنین حالتی گفته می‌شود که ماشین بولتزمان به *تعادل گرمایی*[۱] رسیده است. اگر از یک دمای بالا ماشین بولتزمان شروع به کار کند و به تدریج دمای ماشین کاهش یابد و در یک دمای پایین به تعادل گرمایی برسد، مجموعه‌ای از احتمالات برای حالات مختلف اجزای شبکه به دست می‌آیند، که با تبعیت از آن‌ها انرژی کل سامانه حول مقدار کمینه‌ی سراسری، نوسان خواهد کرد. این فرآیند کاهش دما و تعادل گرمایی به دست آمده در هر مرحله، همان فرآیند شبیه‌ساز سرد کردن فلزات است. در واقع می‌توان ماشین بولتزمان را به صورت یک الگوریتم شبیه‌ساز سرد کردن فلزات دودویی فرض کرد که دارای تابع هدفی به صورت تعریف شده در رابطه (۳-۵) می‌باشد [۲۰، ۲۱].

۳-۳ خلاصه فصل

در این فصل الگوریتم شبیه‌ساز حرارتی به همراه تعدادی از الگوریتم‌های کاهش دما معرفی شدند. سپس کاربرد این روش برای حل مساله فروشنده دوره‌گرد وماشین بولتزمان مطرح شد. در ادامه در فصل آتی الگوریتم بهینه‌سازی مورچه‌ها معرفی خواهد شد.

[۱] thermal equilibrium

فصل چهارم

الگوریتم بهینه‌سازی مورچه‌ها

۴-۱ مقدمه

با وجود آن که فقط ۲ درصد از گونه‌های حشرات دارای زندگی اجتماعی هستند، اما بیش از ۵۰ درصد توده‌ی زیستی حشرات را تشکیل می‌دهند. این میزان در برخی جاها، مانند جنگل‌های بارانی آمازون به بیش از ۷۵ درصد می‌رسد. منظور از زندگی اجتماعی، تجمع تعداد زیادی از یک گونه‌ی خاص در قالب یک *مجموعه* یا *کُلونی*[۱] و تعامل آن‌ها با هم‌دیگر است. همه مورچه‌ها و موریانه‌ها و هم‌چنین برخی از گونه‌های زنبورها در قالب کلونی زندگی می‌کنند. اجتماع حشرات می‌توانند مسایلی را با هم‌کاری یک‌دیگر حل و فصل نمایند که هیچ یک از اعضای آن اجتماع به تنهایی قادر به حل آن‌ها نمی‌باشد. بیشتر این مسایل به صورت مسایل بهینه‌سازی قابل بیان هستند. به عنوان مثال، تلاش حشرات برای یافتن کوتاه‌ترین مسیر در هنگام جستجو برای غذا، تخصیص مناسب نیروهای کاری برای انجام کارهای مختلف و هم‌چنین طبقه‌بندی محل‌های حاوی تخم‌ها و نوزادان از جمله مسایل بهینه‌سازی هستند که حشرات اجتماعی با هم‌کاری یک‌دیگر آن‌ها را حل می‌کنند. این مسایل همگی دارای معادلی در بین مسایل بهینه‌سازی روزمره هستند [۱، ۱۶، ۱۹، ۲۲-۲۶].

هر تلاشی که برای حل یک مساله بهینه‌سازی صورت می‌گیرد، باعث به وجود آمدن اطلاعاتی در مورد آن مساله می‌گردد. به منظور هم‌کاری برای حل یک مساله بهینه‌سازی، داشتن مسیری برای انتقال اطلاعات بین اعضای جامعه، ضروری است. به این ترتیب در هر اجتماع از حشرات، نوع خاصی از ارتباط بین حشرات وجود دارد. این ارتباط در گونه‌های مختلف می‌تواند به صورت مستقیم یا غیرمستقیم در میان حشرات برقرار باشد. به عنوان مثال، هنگامی که یک زنبور عسل یک منبع غذایی

---

[۱] colony

جدید پیدا می‌کند، با اجرای یک رقص ویژه، جهت و فاصله‌ی محل منبع غذایی را به سایر زنبورها اطلاع می‌دهد. این یک ارتباط بسیار مستقیم است. به نحوی که برای آن که زنبوری از پیام مورد نظر اطلاع یابد، می‌بایست رقص زنبور را به طور مستقیم مشاهده و آن را تعبیر و تفسیر کند. ارتباط و تماس فیزیکی نوع دیگری از ارتباط‌های مستقیم میان حشرات اجتماعی است [۱۶، ۲۳].

ارتباط غیر مستقیم نیاز به مهارت بیشتری دارد. در این نوع ارتباط حشره می‌بایست محیط اطراف را به نحوی تغییر دهد که سایر هم‌نوعانش از تغییر محیط آگاه شوند و پیام مورد نظر حشره را دریافت کنند. یکی از بارزترین مثال‌ها برای چنین ارتباطی، ریزش ماده‌ای شیمیایی به نام *فرومون*[۱] به وسیله مورچه‌ها بر روی مسیر حرکت است. به این نحو که مورچه‌ها طی حرکت، دنباله‌ای از فرومون را ترسیم می‌کنند و همواره مشتاق به دنبال کردن مسیرهایی هستند که فرومون بیشتری را داشته باشند. تغییر محیط به منظور ایجاد تغییرات در رفتار از طریق ارتباط به وجود آمده، به نام اصل *اِستیگمِرجی*[۲] معروف است. این اصطلاح برای اولین بار در سال ۱۹۵۹ و به وسیله زیست‌شناس فرانسوی به نام *پیِر پُول گراسه*[۳] برای توضیح رفتار موریانه‌ها به کار برده شده است. این کلمه از ترکیب دو واژه‌ی یونانی *اِستیگما*[۴] به معنی اشاره و *اِرگون*[۵] به معنی کار ساخته شده است. به این ترتیب، استیگمرجی به معنی اشاره (البته غیر مستقیم) برای انجام کاری خواهد بود. استیگمرجی پایه‌ی اصلی بسیاری از حرکت‌های حشرات به خصوص مورچه‌هاست. در جوامع مورچه‌ای، یک مورچه‌ی خاص به نام ملکه وجود دارد که فقط مسئولیت تخم‌گذاری را دارد. در مقابل، سایر مورچه‌ها عملکرد به طور کامل متفاوتی دارند. مورچه‌های یک مجموعه، *خود-ترتیب*[۶] هستند و رفتارهای پیچیده‌ی کل مجموعه فقط ناشی از رفتارهای ساده‌ای است که تک‌تک مورچه‌ها به صورت خود-ترتیب انجام می‌دهند. این خواص جمعی و فردی به نحوی هستند که بر روی مسایل مختلف کارآیی مناسبی دارند. به خصوص، هنگامی که در یک بازه‌ی زمانی خاص، برخی از مورچه‌ها عملکرد مناسبی نداشته باشند، با این وجود عملکرد کلی مجموعه مناسب خواهد بود [۱۶، ۲۲، ۲۳، ۲۵، ۲۶].

عملکرد *مورچه‌های آرژانتینی*[*] در یافتن کوتاه‌ترین مسیر بین لانه و منبع غذایی، بسیار عجیب و حیرت انگیز است. مورچه‌ی آرژانتینی عملا کور است و به طبع کوتاه‌ترین مسیر برای او مفهومی ندارد و به وسیله او قابل شناخت نمی‌باشد. اما با وجود چنین کمبودی، توده‌ای از این مورچه‌ها می‌توانند با هم‌کاری یک‌دیگر، کوتاه‌ترین مسیر موجود بین لانه و محل مواد غذایی را پیدا کنند. برای پی بردن به این خاصیت، آزمایشی در محیطی شبیه به شکل ۴-۱ ترتیب داده شده است. در ابتدا تمامی مورچه‌ها

---

[۱] pheromone

[۲] stigmergy

[۳] Pierre-Paul Grassé

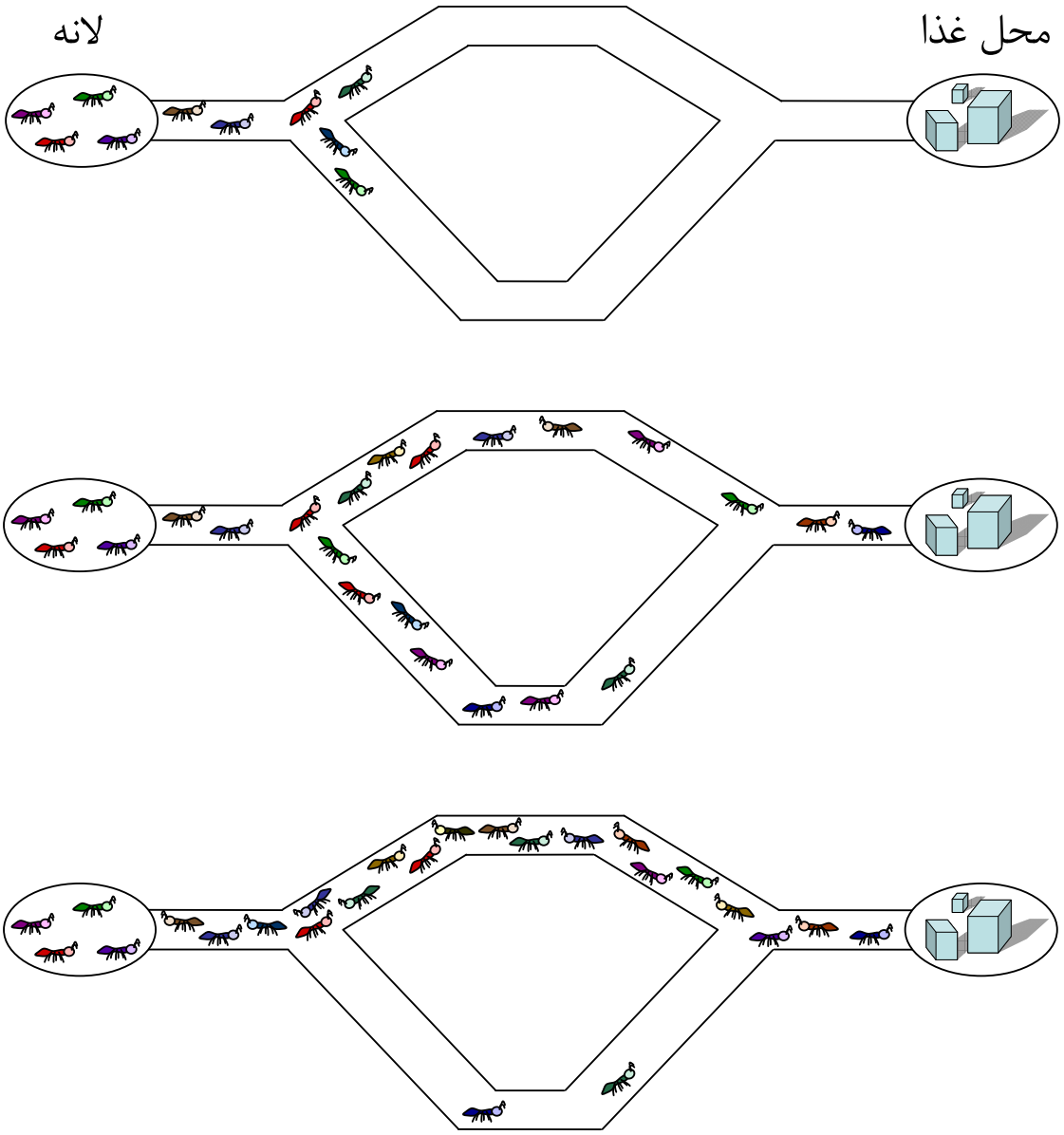
[۴] stigma

[۵] Ergon

[۶] self-organized

[*] نام علمی این گونه‌ی خاص Linepithema Humile است که قبلا به نام Iridomyrmex Humilis نیز خوانده می‌شد. مورچه‌ی آرژانتینی بسیار ریز و تیره رنگ است. زیستگاه این موجود در بخش‌های شمالی آرژانتین، اوروگوئه و پاراگوئه و هم‌چنین بخش‌های جنوبی برزیل است.

در محل لانه هستند و در قبل هیچ مورچه‌ای از مسیر بین لانه و محل مواد غذایی رد نشده است. آزمایش نشان می‌دهد که مورچه‌ها پس از مدتی کوتاه‌ترین مسیر بین لانه و محل مواد غذایی را انتخاب خواهند نمود. این آزمایش به وسیله *گاس*[۱] و همکارانش انجام گرفته است و نتایج آن طی مقالاتی در سال‌های ۱۹۸۹ و ۱۹۹۲ منتشر گردیده است [۱۶، ۲۲، ۲۶].



شکل ۴-۱: رفتار مورچه‌های آرژانتینی در آزمایش گاس و همکارانش [۱].

[۱] goss

۴-۲ الگوریتم بهینه‌سازی کُلونی مورچه‌ها

الگوریتم بهینه‌سازی کُلونی مورچه‌ها، و یا به اختصار الگوریتم مورچه‌ها، از رفتار مورچه‌های طبیعی که در مجموعه‌های بزرگ در کنار هم زندگی می‌کنند الهام گرفته شده است. الگوریتم‌های دیگری نیز بر اساس الگوریتم مورچه‌ها ساخته شده‌اند که همگی *سامانه‌های چند عاملی*[۱] هستند و عامل‌ها مورچه‌های مصنوعی یا به اختصار مورچه‌هایی هستند که مشابه با مورچه‌های واقعی رفتار می‌کنند [۲۲، ۲۶]. الگوریتم مورچه‌ها، یک مثال بارز از هوش جمعی است که در آن عامل‌هایی که قابلیت چندان بالایی ندارند، در کنار هم و با هم‌کاری یک‌دیگر می‌توانند نتایج بسیار خوبی به دست بیاورند. این الگوریتم برای حل و بررسی محدوده‌ی وسیعی از مسایل بهینه‌سازی به کار برده شده است. از این میان می‌توان به حل مساله کلاسیک فروشنده دوره‌گرد و هم‌چنین مساله *راه‌یابی در شبکه‌های مخابرات راه دور*[۲] اشاره نمود [۱۶-۱۸، ۲۶].

۴-۲-۱ الگوریتم ساده شده مورچه‌ها

در این بخش یک الگوریتم بسیار ساده شده برای ارایه‌ی یک الگوی کلی از الگوریتم مورچه‌ها توضیح داده می‌شود. در این الگوریتم، هدف اصلی هر مورچه، یافتن کوتاه‌ترین مسیر بین دو رأس، از گرافی است که مساله مورد بررسی بر روی آن گراف به نحوی مناسب تعریف شده است [۲۴، ۲۶-۲۷].

فرض کنید $G = (V , E)$ یک گراف* معمولی یا غیر وزن‌دار با $n = |V|$ رأس+ باشد. با استفاده از الگوریتم ساده شده مورچه‌ها، می‌توان کوتاه‌ترین مسیر موجود بین دو رأس دلخواه از گراف G را پیدا نمود. به عنوان مثال، یافتن کوتاه‌ترین مسیر موجود بین یک رأس مبدأ به نام s تا یک رأس مقصد به نام d به نحوی که طول مسیر به صورت تعداد یال‌های پشت سر گذاشته شده، تعریف شده باشد. هر یال که رأس iاُم را به رأس jاُم متصل می‌کند به صورت زوج e_{ij} نشان داده می‌شود. برای هر یال به صورت e_{ij} کمیتی به نام دنباله‌ی فرومونی یا میزان فرومون به صورت τ_{ij} در نظر گرفته می‌شود. میزان فرومون به وسیله مورچه‌ها خوانده و یا تغییر داده می‌شود. مقدار و غلظت فرومون موجود بر روی یک یال، به عنوان معیاری برای بررسی مطلوب بودن آن یال، برای انتخاب شدن به وسیله مورچه‌ها جهت ساختن مسیرهای بهتر است. در ابتدای الگوریتم، همه یال‌ها دارای مقداری مساوی از فرومون به اندازه‌ی τ_0 هستند. هر مورچه سیاستی قدم به قدم را برای ایجاد مسیرهای حرکتی به کار می‌گیرد. اطلاعات محلی را، که در هر رأس از گراف و یا در یال‌های خارج شونده از رأس‌ها نگه‌داری می‌شوند به شکلی تصادفی، برای انتخاب مقصد بعدی مورد استفاده قرار می‌گیرند. هنگامی که مورچه‌ی kاُم در

---

[۱] multi-agent systems

[۲] routing in telecommunications network

* $G = (V , E)$ گرافی ساده است که V مجموعه‌ی رأس‌ها و E مجموعه‌ی یال‌های متصل کننده‌ی رأس‌های این گراف هستند.

+ اگر A یک مجموعه باشد، منظور از $|A|$ تعداد اعضای آن مجموعه است.

رأس iاُم قرار دارد، احتمال آن که رأس jاُم را به عنوان مقصد بعدی انتخاب کند به این صورت محاسبه می‌شود:

$$p_{i\to j}^{k} = p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}}{\sum_{m\in\mathcal{N}_i}\tau_{im}} & , j \in \mathcal{N}_i \\ 0 & , j \notin \mathcal{N}_i \end{cases} \qquad (۱-۴)$$

که در آن، $\mathcal{N}_i$ مجموعه رأس‌هایی است که در همسایگی یک قدمی رأس iاُم قرار دارند. به عبارت دیگر، $\mathcal{N}_i$ مجموعه رأس‌هایی از گراف است که به رأس iاُم متصل هستند.

هنگامی که مورچه‌ای از یالی عبور می‌کند، مقداری فرومون نیز در آن یال می‌ریزد. این مقدار در الگوریتم ساده شده، ثابت و به صورت $\Delta\tau$ در نظر گرفته شده است. اگر مورچه‌ای در زمان t از روی یال بین رأس‌های iاُم و jاُم عبور کند، مقدار فرومون آن یال را به صورت زیر تغییر خواهد داد:

$$\tau_{ij}(t) \leftarrow \tau_{ij}(t) + \Delta\tau \qquad (۲-۴)$$

این قانون ساده، رفتار مورچه‌های طبیعی را در ریزش فرومون روی مسیر حرکت شبیه‌سازی می‌کند. مورچه‌ها به این طریق احتمال انتخاب شدن مسیری را که خود طی کرده‌اند، بیشتر می‌کنند و در آینده مورچه‌های دیگر برای انتخاب این مسیر بیشتر رغبت خواهند کرد.

برای جلوگیری از هم‌گرا شدن مورچه‌ها به یک مسیر غیر بهینه، می‌بایست جستجوی خوبی در بین مسیرهای ممکن انجام بگیرد. در همه روش‌های بهینه‌سازی تصادفی، تناقضی بین جستجو[۱] و کاوش[۲] وجود دارد و می‌بایست بین این دو موازنه‌ی مناسبی برقرار شود تا جواب مطلوبی به دست بیاید. در الگوریتم مورچه‌ها برای انجام جستجوی کافی در بین مسیرهای مختلف، مثل فرومون‌های واقعی، فرومون روی یال‌های گراف تبخیر می‌شود. مقدار فرومون یال‌ها به صورت خودکار کاهش پیدا می‌کند تا علاقه‌ی مورچه‌ها برای جستجوی مسیرهای جدید و بهتر، بیشتر شود. فرآیند تبخیر به طور معمول به صورت یک تابع کاهشی ساده در نظر گرفته می‌شود. متداول‌ترین شیوه برای کاهش فرومون، استفاده از تابع نمایی کاهش است. به این ترتیب که مقدار فرومون در هر مرحله، در یک عدد مثبت و کمتر از یک ضرب می‌شود. شکل کلی این عمل به صورت زیر تعریف می‌شود:

$$\tau \leftarrow (1-\rho)\tau \quad , \rho \in (0, 1] \qquad (۳-۴)$$

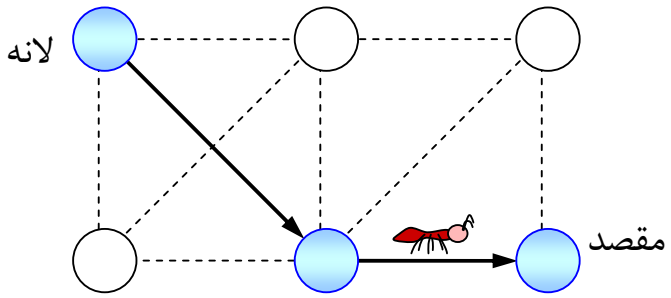
که در آن ρ عددی موسوم به ضریب تبخیر است. در نمونه‌های پیچیده مقدار ρ در مراحل ابتدایی اجرای الگوریتم مقدار به نسبت زیادی است و به مرور کاهش می‌یابد.

---

[۱] exploration

[۲] exploitation

یک آزمایش ساده برای پیدا کردن مسیر بهینه بین لانه و مقصدی فرضی در گرافی ساده که در شکل ۴-۲ نشان داده شده است. این آزمایش، حاکی از عملکرد مناسب الگوریتم ساده شده مورچه‌هاست. هم‌چنین آزمایش‌ها نشان می‌دهند که اگر گراف مساله پیچیده‌تر شود، به عنوان مثال اگر بیش از دو مسیر ممکن بین لانه و مقصد ایجاد شود، رفتار الگوریتم پایداری کمتری خواهد داشت و به شدت به مقادیر پارامترها حساس خواهد بود [۲۶].



شکل ۴-۲ : مسیر بهینه‌ای که به وسیله مورچه‌ها در یک گراف ساده پیدا شده است [۱].

۴-۲-۲ الگوریتم مورچه‌ها

الگوریتم ساده شده مورچه‌ها، به دلیل سادگی بیش از حدش دارای محدودیت‌هایی است. در این بخش، الگوریتم مورچه‌ها در حالت کامل معرفی می‌شود که علاوه بر این که خواص ابتدایی الگوریتم ساده شده را دارد، به منظور برخورد با مسایل پیچیده‌تر، بهینه شده است و محدودیت‌های موجود در الگوریتم ساده شده، در آن وجود ندارند. به عنوان مثال، مقدار فرومونی که به وسیله هر مورچه روی یال‌ها ریخته می‌شود، متناسب با کیفیت جواب‌هایی است که مورچه یافته است. لذا اطلاعاتی که به وسیله فرومون‌ها به وجود می‌آید به مراتب کارآتر است و تاثیر بهتری در جهت‌دهی مناسب برای جستجو به وسیله مورچه‌ها دارد. از طرفی الگوریتم ساده‌ی مورچه‌ها، برای برخورد با مسایل بزرگ‌تر و پیچیده‌تر مناسب نمی‌باشد و فقط برای پیدا کردن کوتاه‌ترین مسیر بین دو رأس از یک گراف، قابل استفاده است. اما برای یافتن کوتاه‌ترین مسیر *هامیلتونی*[*] در یک گراف مناسب نیست و برای حل چنین مساله‌ای، می‌بایست مورچه‌ها دارای نوعی از حافظه نیز باشند [۱۱، ۱۶-۱۹، ۲۲، ۲۴-۲۷].

الگوریتم مورچه‌ها و الگوریتم‌های برگرفته از آن می‌توانند برای حل دسته‌ای از مسایل بهینه‌سازی گسسته مورد استفاده قرار گیرند. همان طور که در فصل اول گفته شد، در مسایل بهینه‌سازی گسسته، مجموعه مقادیری که متغیرها به خود می‌گیرند، یک مجموعه شمارا و اغلب متناهی است. اما آن دسته از مسایل گسسته که الگوریتم‌های مورچه برای حل آن‌ها مناسب هستند، دارای خواص مشخصی هستند. این خواص عبارتند از [۲۲، ۲۶].

[*] مسیر هامیلتونی، مسیر بسته‌ای در یک گراف است به نحوی که هر رأس از گراف را یک و فقط یک بار در بر دارد.

- یک مجموعه متناهی از عناصر به صورت $C = \{c_1, c_2, \ldots, c_{N_C}\}$ داده شده باشد.

- مجموعه‌ای متناهی از ارتباط‌ها یا انتقال‌های ممکن بین اعضای مجموعه C به صورت $L = \{l_{c_i c_j} \mid (c_i, c_j) \in \tilde{C} \subseteq C \times C\}$ تعریف شده باشد، که در آن $\tilde{C}$ زیرمجموعه‌ای از *ضرب کارتزینی*[۱] $C \times C$ است*. درضمن رابطه‌ی $|L| \le N_C^2$ نیز همواره برقرار خواهد بود.

- برای هر ارتباط به صورت $l_{c_i c_j} \in L$ ، یک تابع هزینه‌ی ارتباط که به زمان نیز ممکن است وابسته باشد به صورت $J_{c_i c_j} = J(l_{c_i c_j}, t)$ تعریف شده باشد.

- مجموعه‌ای متناهی از قیود به شکل $\Omega = \Omega(C, L, t)$ تعریف شده باشد. این قیود بر روی اعضای مجموعه‌های C و L تعریف شده‌اند و متغیر t نشان دهنده‌ی وابستگی احتمالی این قیود به زمان هستند.

- حالات مساله به صورت دنباله‌هایی از اعضای C و یا L تعریف شده باشند. به عنوان یک نمونه، $s = \langle c_i, c_j, \ldots, c_k, \ldots \rangle$ یک دنباله است که روی اعضای C تعریف شده است و یک حالت نامیده می‌شود. فرض کنید که S مجموعه تمام حالات قابل تعریف باشد. مجموعه حالاتی که قیود $\Omega(C, L, t)$ را برآورده می‌کنند، زیرمجموعه‌ای از S به صورت $\tilde{S}$ خواهد بود. اعضای مجموعه $\tilde{S}$، *حالات شدنی*[۲] را برای مساله تعریف می‌کنند. طول یک دنباله به نام s که تعداد اعضای آن دنباله است، به صورت $|s|$ نمایش داده می‌شود.

- یک ساختار همسایگی به صورت واقعی قابل تعریف باشد. حالت s_2 همسایه یا مجاور حالت s_1 شناخته می‌شود اگر:

(۱) s_1 و s_2 هر دو در مجموعه حالات S باشند.

(۲) بتوان با یک حرکت حالت s_1 را به حالت s_2 تبدیل نمود. به عبارت دیگر اگر c_1 آخرین عضو دنباله‌ی توصیف کننده‌ی s_1 باشد، می‌باید عضوی از C به نام c_2 وجود داشته باشد، به نحوی که مسیری بین c_1 و c_2 وجود داشته (یعنی $l_{c_1 c_2} \in L$) و هم‌چنین $s_2 \equiv \langle s_1, c_2 \rangle$ باشد.

- پاسخ مساله به صورت عضوی از $\tilde{S}$ قابل تعریف باشد. یک پاسخ، چند بعدی خوانده می‌شود اگر به صورت مجموعه‌ای از دنباله‌های متمایز روی اعضای C تعریف شده باشد. توجه شود که منظور از پاسخ، پاسخ بهینه نیست.

---

[۱] cartesian product

* ضرب کاتزینی دو مجموعه‌ی A و B به صورت $A \times B \triangleq \{(a, b) \mid a \in A, b \in B\}$ تعریف می‌شود.

[۲] feasible states

- متناظر با هر پاسخ به صورت Ψ، هزینه‌ای به صورت $J_{\Psi}(L,t)$ تعریف شده باشد. این هزینه، تابعی از هزینه‌ی تمام ارتباط‌های تشکیل دهنده‌ی Ψ می‌باشد. متغیر t نیز، طبق معمول، نشان دهنده‌ی وابستگی احتمالی این هزینه به زمان می‌باشد.

فرض کنید، $G=(C,L)$ گرافی متناظر با مساله بهنیه‌سازی گسسته‌ای باشد که به صورت فوق تعریف شد. پاسخ‌های این مساله بهینه‌سازی به صورت مسیرهای شدنی در این گراف، قابل بیان هستند. الگوریتم‌های مورچه می‌توانند برای یافتن پاسخ‌هایی که قیود Ω را برآورده کنند، به کار برده شوند. در ادامه، هر کدام از اعضای C یک گره یا رأس نامیده می‌شود و هم‌چنین یک عضو مجموعه L نیز، ارتباط و یا یال خوانده می‌شود.

در الگوریتم مورچه‌ها، یک مجموعه یا کلونی از مورچه‌ها، در کنار هم مساله بهینه‌سازی گسسته‌ای را، با استفاده از تعریف آن که به شکل یک گراف ارایه شده است، حل می‌کنند. اطلاعاتی که مورچه‌ها در هنگام جستجو به دست می‌آورند، در قالب مقادیر فرومون به صورت τ_{ij} روی یال $l_{ij}=l_{c_ic_j}$ ذخیره می‌شوند. در واقع مقادیر فرومون یک حافظه‌ی بلند مدت برای ذخیره‌ی اطلاعات همه مورچه‌ها است. بسته به تعریف مساله، دنباله‌های فرومونی، می‌توانند روی همه یا تعداد خاصی از یال‌های گراف مساله ایجاد شوند. در برخی مسایل، اطلاعاتی اولیه به صورت کمیت‌های مرتبط با یال‌ها تعریف می‌شوند. این کمیت‌ها که *ارزش‌های ذهنی* یا *غیرمستدل*[۱] نامیده می‌شوند، برای یال l_{ij} به صورت η_{ij} تعریف می‌شود. این اطلاعات به وسیله مورچه‌ها استفاده می‌شوند، اما منبع ایجاد آن‌ها خود مورچه‌ها نیستند. بلکه یک عامل خارجی یا همان طراح مساله، این اطلاعات را برای استفاده‌ی مورچه‌ها در اختیار آن‌ها قرار می‌دهد.

کلونی مورچه‌ها دارای یک سری خواص عمومی است که مهم‌ترین آن‌ها عبارتند از [۲۲، ۲۶]:

- اگر چه هر کدام از مورچه‌ها به حد کافی برای یافتن یک راه حل برای مساله، پیچیده هستند و به احتمال راه حل خوبی نیز پیدا خواهند نمود، اما راه حل‌هایی که دارای کیفیت مطلوب باشند، فقط از طریق هم‌کاری و تعامل بین مورچه‌ها قابل دسترسی است.

- هر مورچه فقط از اطلاعات فردی خود و یا از اطلاعات محلی رأس یا گرهی که در آن قرار دارد بهره می‌برد.

- مورچه‌ها به صورت غیر مستقیم با یک‌دیگر در ارتباط هستند. این ارتباط از طریق تغییراتی است که آن‌ها در مقادیر فرومون مسیرهای مختلف ایجاد می‌کنند.

---

[۱] heuristic values

هم‌چنین مورچه‌های موجود در کلونی، دارای خواصی هستند، که عبارتند از [۲۲، ۲۶]:

- یک مورچه، به دنبال راه حلی می‌گردد که کمترین هزینه را در بر داشته باشد. این هزینه به صورت $J^* = \min_{\Psi} J_{\Psi}(L, t)$ قابل تعریف است.
- مورچه‌ی kاُم دارای یک حافظه‌ی شخصی به نام $\mathcal{M}^k$ است که مسیری را که مورچه طی کرده است در آن ذخیره می‌شود. حافظه می‌تواند برای (الف) ایجاد راه حل‌های مجاز و شدنی، (ب) ارزیابی راه حل‌های یافته شده و (پ) بازگشت مسیر به صورت معکوس، استفاده می‌شود.
- مورچه‌ی kاُم که در حالت $s_r = \langle s_{r-1}, i \rangle$ قرار دارد، می‌تواند به هر رأس مانند j برود که عضوی از مجموعه $\mathcal{N}_i^k \triangleq \left\{ j \,\middle|\, (j \in \mathcal{N}_i) \text{and} \left(\langle s_r, j \rangle \in \tilde{S} \right) \right\}$ است. $\mathcal{N}_i$ مجموعه رأس‌هایی است که در هم‌سایگی رأس iاُم قرار دارد و به صورت $\mathcal{N}_i \triangleq \left\{ j \,\middle|\, l_{ij} \in L \right\}$ تعریف می‌شود.
- مورچه‌ی kاُم از یک حالت اولیه‌ی s_o^k شروع به کار می‌کند.
- برای مورچه‌ی kاُم مجموعه شرایط خاتمه‌ی مخصوصی به صورت e^k دارد.
- مورچه‌ها از حالت اولیه‌ی خودشان شروع به حرکت می‌کنند. در هر حرکت به یکی از حالت‌های مجاور که شدنی و مجاز باشد، می‌روند. این کار باعث به وجود آمدن یک راه حل شدنی برای مساله می‌شود. حرکت یک مورچه تا جایی ادامه می‌یابد که حداقل یکی از شرایط خاتمه‌ی e^k برآورده شود.
- مورچه‌ی kاُم که در رأس iاُم قرار دارد، می‌تواند به یکی از رأس‌هایی که در $\mathcal{N}_i^k$ قرار دارد، حرکت کند. انتخاب مقصد، با توجه به یک قانون احتمالی مشخص انجام می‌گیرد.
- قانون احتمالی مورچه‌ها تابعی از (الف) قیود مساله، (ب) حافظه‌ی هر مورچه و (پ) اطلاعات محلی ذخیره شده در هر رأس است. اطلاعات محلی هر رأس از ترکیب اطلاعات فرومونی τ_{ij} و اطلاعات ذهنی η_{ij} به وجود می‌آید. این اطلاعات به صورت سازمان‌یافته در جدولی موسوم به *جدول مسیریابی*[۱] ذخیره می‌گردند.
- هر مورچه، در هنگام حرکت از رأس iاُم به سمت رأس jاُم، مقدار فرومون τ_{ij} روی یال l_{ij} را تغییر می‌دهد. این کار *تجدید فرومون گام به گام و آنی*[۲] نامیده می‌شود.

---

[۱] routing table

[۲] online step-by-step pheromone update

- اگر مورچه‌ای مسیری را که یافته رو به عقب برگردد و فرومون یال‌های تشکیل دهنده‌ی آن را عوض کند، این عمل *تجدید تاخیری فرومون*[۱] نامیده می‌شود.

به زبان ساده‌تر، می‌توان فعالیت مورچه‌ها را به این صورت جمع‌بندی کرد. مجموعه‌ای از مورچه‌ها به صورت هم‌زمان اما مستقل و غیر هماهنگ روی گراف مربوط به مساله حرکت می‌کنند. نحوه‌ی حرکت مورچه‌ها به وسیله یک قانون تصادفی معین، مشخص می‌شود که از روی اطلاعات محلی ذخیره شده در رأس‌های گراف ایجاد شده است. مورچه‌ها با حرکت‌های خود، راه حل‌هایی را برای مساله بهینه‌سازی می‌سازند. هنگامی که راه حلی به وسیله یک مورچه ساخته شد و یا در حین ساخته شدن راه حل به وسیله مورچه ارزیابی می‌شود و مورچه کیفیت جواب خود را با ریزش فرومون روی مسیرش، به سایرین خبر می‌دهد. اطلاعاتی که در فرومون‌ها وجود دارد، جستجوی مورچه‌های بعدی را جهت دهی خواهد کرد.

در کنار فعالیت جمعی مورچه‌ها برای یافتن جواب بهینه، الگوریتم مورچه‌ها نیز می‌تواند شامل دو عمل اضافی دیگر باشد: *تبخیر فرومون*[۲] و *کارهای کمکی*[۳]. تبخیر فرومون باعث می‌شود که میزان فرومون یال‌های گراف با سپری شدن زمان به صورت خودکار کمتر شود. این عمل باعث می‌شود که از هم‌گرایی سریع الگوریتم به یک جواب نامناسب، جلوگیری به عمل آید. در واقع تبخیر فرومون، به نوعی *فراموش‌کاری* مورچه‌ها را پیاده‌سازی می‌کند و باعث می‌شود که آن‌ها نواحی جدیدی را برای یافتن جواب‌های بهینه‌تر جستجو کنند. کارهای کمکی، کارهای متمرکزی هستند که به وسیله هیچ یک از مورچه‌ها قابل انجام نمی‌باشند. به عنوان مثال، استفاده از یک الگوریتم بهینه‌سازی محلی دیگر برای بهتر کردن نسبی نتیجه، از جمله کارهای کمکی است. مثال دیگر، ریزش مقداری فرومون اضافه بر روی کوتاه‌ترین مسیر یافته شده به وسیله مورچه‌ها است. تغییر مقدار فرومون یال‌ها، به وسیله کارهای کمکی را *تجدید فرومون غیر آنی* یا *خارج از خط*[۴] می‌نامند. فهرست عملیاتی در یک مرحله از مورچه‌ها انجام می‌شود، در شکل ۴-۳ به صورت شبه-برنامه آمده است.

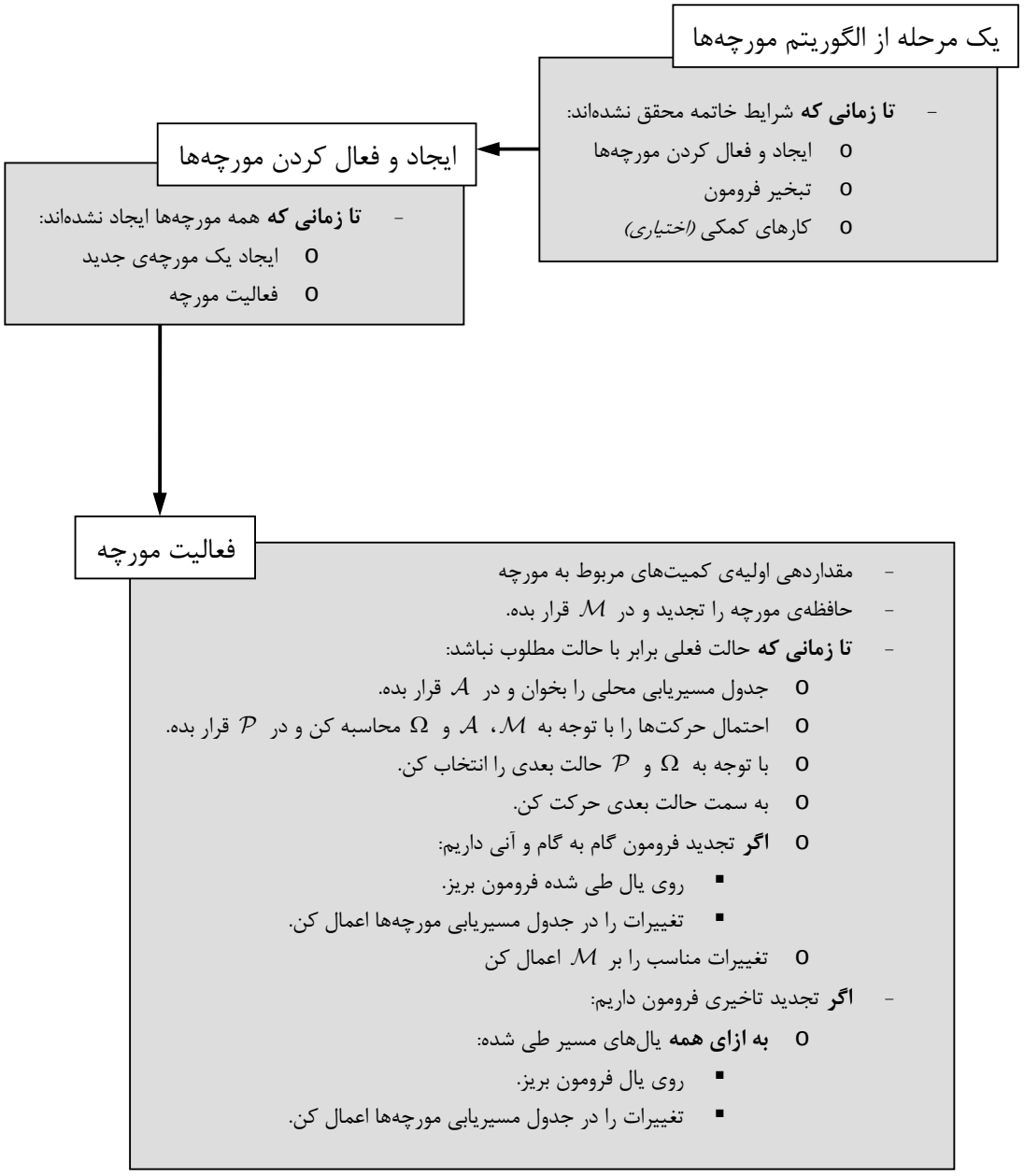
---

[۱] delayed pheromone update

[۲] pheromone evaporation

[۳] daemon actions

[۴] offline pheromone update



شکل ۴-۳ : فهرست عملیاتی که در یک مرحله از اجرای الگوریتم مورچه‌ها انجام می‌شوند.

۴-۳ الگوریتم مورچه‌ها برای مساله فروشنده دوره‌گرد[۱]

مساله فروشنده دوره‌گرد یا TSP، که در فصل سوم معرفی شد، یک نمونه از مسایلی است که می‌توان با استفاده از الگوریتم مورچه‌ها به حل آن پرداخت. منظور از حل مساله TSP، پیدا کردن کوتاه‌ترین مسیری است که مجموعه تمام نقاط $v \in V$ را به هم‌دیگر متصل می‌کند. این نقاط رأس‌های گرافی کامل و وزن‌دار به صورت $G(V, V \times V, d)$ می‌باشند، که در آن $d: V \times V \to \mathbb{R}^{\geq 0}$، یک *اندازه* یا *متر* بر روی $V \times V$ می‌باشد و نشان دهنده‌ی فاصله‌ی بین نقاط است. در واقع مساله فروشنده دوره‌گرد، تعمیم مستقیمی از نحوه‌ی رفتار مورچه‌ها برای پیدا کردن کوتاه‌ترین مسیر بین لانه و محل غذا می‌باشد. نقاط مجموعه V، شهر و مسیر بین هر دو شهر راه نامیده می‌شود.

الگوریتم مورچه‌ها، مساله فروشنده دوره‌گرد را به صورت مرحله به مرحله و تکراری حل می‌کند [۲۲، ۲۶، ۲۸، ۳۰]. در هر تکرار، مورچه‌ها از هر شهر به شهرهای ملاقات نشده حرکت می‌کنند تا آن که همه شهرها را پشت سر بگذارند. مورچه‌ها با توجه به یک قانون احتمالی معین، مقصد بعدی را برای حرکت‌شان تعیین می‌کنند. برای این کار از اطلاعات فرومونی τ_{ij} و اطلاعات ذهنی η_{ij} بهره می‌گیرند. τ_{ij} معیاری بر اساس سابقه‌ی هر حرکت در گذشته است و در مقابل η_{ij} یک معیاری فوری‌تر برای انتخاب یک حرکت می‌باشد. هر چه مقدار τ_{ij} و η_{ij} بیشتر باشد، یال متناظر با آن‌ها بیشتر مورد توجه مورچه‌ها خواهد بود. در مساله فروشنده دوره‌گرد، اطلاعات ذهنی به صورت $\eta_{ij} = 1/d_{ij}$ تعریف می‌شود که در آن d_{ij} فاصله‌ی بین شهر iاُم و jاُم می‌باشد. به این ترتیب مسیرهایی که دارای طول کمتری هستند، بیشتر ترجیح داده می‌شوند. بر خلاف مورچه‌های واقعی، مورچه‌های الگوریتم پس از طی شدن مسیر، بر اساس وضعیت نسبی مسیرشان در مقابل مسیر سایر مورچه‌ها، بر روی مسیرشان فرومون می‌ریزند.

اگر مورچه‌ی kاُم در شهر iاُم باشد، احتمال انتخاب شهر jاُم به عنوان مقصد بعدی از روی رابطه‌ی (۴-۴) تعیین می‌شود. $\mathcal{N}_i^k$ مجموعه شهرهایی هستند که در همسایگی شهر iاُم قرار دارند و مورچه‌ی kاُم از آن‌ها عبور نکرده است.

$$p_{i\to j}^k = p_{ij}^k = \begin{cases} \dfrac{(\tau_{ij})^\alpha (\eta_{ij})^\beta}{\sum_{m \in \mathcal{N}_i^k} (\tau_{im})^\alpha (\eta_{im})^\beta} & , j \in \mathcal{N}_i^k \\ 0 & , j \notin \mathcal{N}_i^k \end{cases} \qquad (۴-۴)$$

در رابطه‌ی (۴-۴) α و β اعدادی ثابت و مثبت هستند که برای وزن‌دهی اطلاعات فرومون و اطلاعات ذهنی به کار می‌روند. هر چه قدر وزن یک نوع از اطلاعات بیشتر باشد، تاثیر بیشتری در نحوه‌ی تصمیم‌گیری مورچه‌ها و در جواب به دست آمده به وسیله آن‌ها دارد. برای بسیاری از مسایل،

---

[۱] Traveling Salesman Problem (TSP)

$\alpha = \beta = 1$ به طور معمول نتایج خوبی به همراه دارد. اما برای برخی از مسایل، می‌توان این دو ضریب را به نحوی تغییر داد که نتایج بهتری به دست بیاید [۲۶].

در مساله فروشنده دوره‌گرد، اگر $\alpha = 0$ باشد، احتمال انتخاب شهرهای نزدیک‌تر بیشتر است. در واقع در این حالت الگوریتم مورچه‌ها به یک الگوریتم جستجوی تصادفی و البته حریصانه تبدیل می‌شود. در مقابل اگر $\beta = 0$ باشد، فقط از اطلاعات فرومونی استفاده می‌شود. در این حالت هم‌گرایی سریع‌تری وجود خواهد داشت و به عبارتی باعث ایجاد رکود[۱] در روند الگوریتم خواهد شد. اگر پس از مدتی همه مورچه‌ها به یک مسیر واحد که به طور معمول نامناسب نیز می‌باشد، هم‌گرا شوند، گفته می‌شود که الگوریتم به رکود رسیده است. به این ترتیب می‌بایست موازنه‌ای مناسب میان اطلاعات فرومونی و اطلاعات ذهنی به وجود بیاید که این موازنه از طریق انتخاب مقادیر مناسب برای α و β قابل دسترسی است [۱۶، ۲۶].

نشان داده شده است که برای مساله فروشنده دوره‌گرد، انتخاب $\beta > 1$ نتایج خوبی را در بر دارد. به عنوان مثال $\beta = 2$ در [۲۸، ۳۲] و $\beta = 5$ در [۲۹، ۳۰] مورد استفاده قرار گرفته‌اند و نتایج خوبی به دست آمده‌اند. هم‌چنین در [۳۴] نشان داده شده است که انتخاب $\alpha > 1$ باعث هم‌گرایی سریع‌تر الگوریتم می‌شود که البته کیفیت جواب نهایی، با افزایش سرعت هم‌گرایی الگوریتم کمتر می‌شود.

پس از آن که مورچه‌ها همگی مسیرهای مورد نظر را به طور کامل طی کردند، مسیرهای طی شده به وسیله آن‌ها ارزیابی می‌شود. طول مسیر به عنوان معیاری برای ارزیابی جواب‌های به دست آمده به وسیله مورچه‌ها مورد استفاده قرار می‌گیرد. فرومون همه یال‌هایی که در مسیر هر مورچه بوده‌اند در تکرار tاُم از الگوریتم، به صورت زیر تغییر داده می‌شود:

$$\tau_{ij}(t) \leftarrow \tau_{ij}(t) + \Delta\tau_{ij}^{k}(t) \qquad (۵-۴)$$

در این رابطه $\Delta\tau_{ij}^{k}(t)$ مقدار فرومونی است که به وسیله مورچه‌ی kاُم به یال l_{ij} از گراف افزوده می‌شود. اگر $\Psi^{k}(t)$ مسیری باشد که مورچه‌ی kاُم در تکرار tاُم طی کرده است، طول مسیر Ψ^{k} از روی تابع هزینه‌ای که در مساله تعریف شده است به صورت $J_{\Psi}^{k} = J(\Psi^{k})$ قابل محاسبه است. در ادامه، برای اختصار، از نوشتن متغیر زمانی t در روابط، خودداری شده است. به طور معمول میزان افزایش فرومون $\Delta\tau_{ij}^{k}$ به صورت زیر در نظر گرفته می‌شود:

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{Q}{J_{\Psi}^{k}} & , l_{ij} \in \Psi^{k} \\ 0 & , l_{ij} \notin \Psi^{k} \end{cases} \qquad (۶-۴)$$

که در آن Q مقداری ثابت است و در بیشتر مواقع برابر با یک در نظر گرفته می‌شود [۱۶، ۱۸، ۲۲].

---

[۱] stagnation

روش معمول دیگری که برای تعیین مقدار $\Delta\tau_{ij}^{k}$ وجود دارد، به این صورت است. فرض کنید $\Psi^{+}(t)$ مسیر بهینه‌ای باشد که مورچه‌ها در تکرار tاُم می‌یابند و $\Psi^{*}(t)$ بهترین جوابی باشد که در تکرار tاُم و تکرارهای قبلی یافته شده است. در این حالت، $\Delta\tau^{k}$ به صورت زیر محاسبه می‌شود [۱۶، ۱۸، ۲۲]:

$$\Delta\tau_{ij}^{k} = \begin{cases} \Delta/2 & , \Psi^{k} = \Psi^{+} \\ \Delta/2 & , \Psi^{k} = \Psi^{*} \\ 0 & , \text{otherwise} \end{cases} \qquad (۷-۴)$$

که در آن Δ مقداری ثابت است و به طور معمول از روی رابطه‌ی زیر تعیین می‌شود:

$$\Delta = \rho.\tau_{0} \qquad (۸-۴)$$

که در آن τ_{0} مقدار اولیه‌ی فرومونی است که بر روی همه یال‌ها قرار دارد و به صورت یک عدد مثبت و کوچک در نظر گرفته می‌شود. هم‌چنین ρ ضریب تبخیر می‌باشد که عددی در بازه‌ی $(0, 1]$ است.

پس از آن که مقادیر فرومون روی مسیرها تغییر داده شدند، فرومون روی یال‌های مختلف گراف تبخیر می‌شوند. این رابطه برای همه یال‌های گراف مساله اجرا می‌شود:

$$\tau_{ij}(t) \leftarrow (1-\rho)\tau_{ij}(t) \quad , \rho \in (0, 1] \qquad (۹-۴)$$

مقادیر کوچک برای ضریب تبخیر ρ، منجر به هم‌گرایی سریع می‌شوند. یک مقدار کوچک برای ضریب تبخیر، باعث می‌شود که مورچه‌ها علاقه‌ای به گردش در یال‌های مختلف گراف نداشته باشند و جواب‌های اولیه‌شان اکتفا کنند. در مقابل اگر $\rho = 1$ در نظر گرفته شود، روند هم‌گرایی الگوریتم بسیار کند می‌شود و در بسیاری از موارد، در اصل الگوریتم هم‌گرا نمی‌شود. در واقع ضریب تبخیر به طور مستقیم، زمان هم‌گرایی الگوریتم را تحت تاثیر قرار می‌دهد. در نتیجه، ضریب تبخیر ρ می‌بایستی با توجه به مهلتی که برای اجرای الگوریتم وجود دارد، تعیین شود. در واقع، در مورد الگوریتم مورچه‌ها، مقدار ρ، اهرمی است که به واسطه‌ی آن می‌توان تعادلی بین جستجو و بهره برداری را بر قرار نمود. در مجموع، کیفیت جوابی که خواسته شده است، بر تعیین مقدار ρ بیشترین تاثیر را دارد [۱۶، ۱۹، ۲۶، ۳۴].

۴-۴ برخی از نسخه‌های تغییر یافته‌ی الگوریتم مورچه‌ها

اولین نسخه‌ی الگوریتم مورچه‌ها با نام *سامانه مورچه‌ای*[۱] (AS) در سال ۱۹۹۲ و به وسیله *مارکو دوریگو*[۲] در رساله‌ی دکتری معرفی شد. سامانه مورچه‌ای، متشکل از ۳ نوع الگوریتم مورچه بود که فقط در نحوه‌ی تغییر داده شدن فرومون‌ها با هم تفاوت داشتند. اسامی این سه نوع الگوریتم، *مورچه-چگالی*[۳]، *مورچه-مقدار*[۴] و *مورچه-دوره*[۵] بودند. از میان این سه نوع الگوریتم، عملکرد الگوریتم مورچه-دوره به مراتب بهتر از دو الگوریتم دیگر است. لذا اغلب وقتی صحبت از سامانه مورچه‌ای به میان می‌آید، منظور همان الگوریتم مورچه-دوره است. الگوریتم‌های زیادی با الهام گرفتن از الگوریتم مورچه-دوره ایجاد شده‌اند که کاربردهای موفقیت‌آمیز و فراوانی در مسایل مختلف دارند. در این متن، منظور از عبارت *الگوریتم مورچه‌ها*، همان الگوریتم مورچه‌ی ابتدایی یا AS است [۱۶، ۱۹، ۲۲، ۲۴، ۲۶].

۴-۴-۱ الگوریتم مورچه‌ها و نخبه‌گرایی

نسخه‌ی تغییر یافته‌ای از سامانه مورچه‌ای در سال ۱۹۹۶ به وسیله دوریگو و همکارانش معرفی شد که مورچه‌های نخبه‌گرا را در بر داشت. در این نسخه، مورچه‌ای که بهترین راه‌حل و یا به عبارت بهتر، کوتاه‌ترین مسیر را یافته باشد، فرومون بیشتری بر روی گراف می‌ریزد. این عمل باعث می‌شود که سایر مورچه‌های به بهترین راه حل موجود نزدیک‌تر شوند. نحوه‌ی فرومون ریزی که در رابطه‌ی (۴-۷) آمده است، نوعی از الگوریتم مورچه‌ی نخبه‌گرا را ایجاد می‌کند [۱۶، ۱۹، ۳۴].

۴-۴-۲ الگوریتم مورچه‌ها و یادگیری تقویتی[۶]

یکی از مسایل مهمی که در بحث *یادگیری ماشینی*[۷] مطرح می‌شود، مساله یادگیری تقویتی است که نوع خاصی از *یادگیری غیر نظارت شده*[۸] است. در یادگیری تقویتی، هدف تعلیم دادن یک عامل یادگیرنده است که فقط باید با تکیه بر نتیجه‌ی کارهایی که انجام می‌دهد، نحوه‌ی عملکرد صحیح در محیط را یاد بگیرد. عامل یادگیرنده، کاری را انجام می‌دهد و تغییراتی احتمالی در محیط اطراف به وجود می‌آورد. از طرف محیط، حالت جدید محیط به همراه نتیجه‌ای عددی از رفتار عامل، به عامل برگشت داده می‌شود. عامل وظیفه دارد با بررسی ورودی‌هایی که از طرف محیط وارد می‌شوند و سابقه‌ای که درمورد کارهای قبلی دارد، نحوه‌ی عملکرد صحیح را یاد بگیرد [۱۹، ۲۸، ۲۹].

---

[۱] Ant System(AS)
[۲] Marco dorigo
[۳] ant-density
[۴] ant-quantity
[۵] ant-cycle
[۶] reinforcement learning
[۷] machine learning
[۸] unsupervised learning

یکی از روش‌هایی که برای حل مسایل یادگیری تقویتی استفاده می‌شود، روش Q-Learning یا QL است. در مسایل یادگیری تقویتی، عامل در شرایط و حالات مختلف قرار می‌گیرد و می‌تواند از بین چندین عمل، عملی را انتخاب کند. در روش QL معیاری که برای ارزیابی اعمال مختلف استفاده می‌شود، مقداری عددی است که به یک زوج مرتب از حالت و عمل نسبت داده می‌شود. این مقدار که *تابع ارزش حالت-عمل*[۱] خوانده می‌شود، معیاری برای انتخاب یک عمل از بین چندین عمل قابل انتخاب می‌باشد. مقادیر ارزش حالت-عمل در طی اجرای الگوریتم، به نحو خاصی تغییر داده می‌شوند که عامل بتواند به نتیجه‌ی مطلوب برسد. در حل مساله فروشنده دوره‌گرد به وسیله الگوریتم مورچه‌ها، در هر شهر (حالت) از بین حرکت‌های مختلف (اعمال مختلف) یک مورد انتخاب می‌شود. این انتخاب بر اساس اطلاعات فرومونی و اطلاعات ذهنی انجام می‌گیرد. اطلاعات فرومونی و اطلاعات ذهنی، همانند ارزش حالت-عمل هستند. در سال ۱۹۹۵ دوریگو و *گامباردلا*[۲] با الهام از شیوه‌ی تغییر مقادیر ارزش حالت-عمل در الگوریتم QL، الگوریتم جدیدی بر اساس الگوریتم مورچه‌ها به نام Ant-Q ایجاد کردند [۲۷]. البته این الگوریتم برتری خاصی نسبت به الگوریتم مورچه‌ها ندارد و در واقع نسخه‌ای اولیه از الگوریتمی است که در بند بعدی توضیح داده می‌شود [۱۹، ۲۸].

۴-۴-۳ سامانه کلونی مورچه‌ها[۳]

الگوریتم سامانه کلونی مورچه‌ها یا ACS در سال ۱۹۹۷ به وسیله دوریگو و گامباردلا برای بهینه کردن عملکرد الگوریتم مورچه‌ها، در حل مسایل پیچیده‌تر با ابعاد بیشتر، طراحی شده است. این الگوریتم بر اساس تغییراتی که در الگوریتم مورچه‌های اولیه ایجاد شده است، به وجود آمده است [۱۹، ۲۶-۲۸]. این تغییرها، بیشتر به منظور ایجاد تعادل میان جستجو و بهره‌برداری ایجاد شده‌اند. تغییرهای اعمال شده بر الگوریتم مورچه‌ها برای ایجاد ACS به صورت زیر است:

الف) قاعده‌ی انتخاب مقصد: در الگوریتم ACS نحوه‌ی انتخاب مقصد متفاوت با AS است. با فرض این که q_0 عددی در بازه‌ی $[0, 1]$ باشد، با احتمال q_0، مسیری برای حرکت انتخاب می‌شود که دارای بیشترین مقدار فرومون و کم‌ترین فاصله باشد. با احتمال $1 - q_0$، مسیر حرکت مانند الگوریتم AS انتخاب می‌گردد. در واقع q_0 تعادلی نیز میان جستجو و بهره‌برداری برقرار می‌کند. جستجوی بیشتر باعث افزایش *گوناگونی*[۴] جواب‌ها و بهره‌برداری باعث *تاکید*[۵] بر بهترین جواب یافته شده می‌شود. مقصد مقصد بعدی مورچه‌ی kاُم که در شهر iاُم می‌باشد، به صورت زیر تعیین می‌شود:

---

[۱] sate-action value function

[۲] gambardella

[۳] Ant Colony System (ACS)

[۴] diversification

[۵] intensification

$$j = \begin{cases} \underset{m \in \mathcal{N}_i^k}{\operatorname{argmax}} \left[\left(\tau_{im}^k \right)^{\alpha} \left(\eta_{im}^k \right)^{\beta} \right] & , q \le q_0 \\ j_0 & , q > q_0 \end{cases} \qquad (۱۰-۴)$$

که در آن q عددی تصادفی در بازه‌ی $[0, 1]$ و با توزیع یکنواخت می‌باشد. j_0 نیز مقصدی است تصادفی که با توجه به احتمال تعریف شده در رابطه‌ی (۴-۴) انتخاب می‌گردد.

ب) نحوه‌ی تجدید فرومون: در الگوریتم ACS دو نوع تجدید فرومون اعمال می‌شود. نوعی از تجدید فرومون محلی است، که مورچه‌ها به هنگام حرکت در مسیر، بر روی یال‌هایی که از آن‌ها عبور می‌کنند، فرومون می‌ریزند. این نوع از فرومون‌ریزی به صورت زیر انجام می‌گیرد:

$$\tau_{ij} \leftarrow \tau_{ij} + \rho \tau_0 \qquad (۱۱-۴)$$

که در آن τ_0 مقدار فرومون اولیه‌ی موجود بر روی مسیرهاست و ρ نیز ضریب تبخیر می‌باشد. نوع دیگری از فرومون‌ریزی که سراسری است، فقط بر روی بهترین مسیر یافته شده در هر تکرار، یعنی Ψ^+ اعمال می‌گردد. اگر $l_{ij} \in \Psi^+$ باشد، و J^+ طول یا هزینه‌ی متناظر با مسیر Ψ^+ باشد، فرومون یال l_{ij} به این صورت تغییر می‌یابد:

$$\tau_{ij} \leftarrow (1 - \rho).\tau_{ij} + \frac{\rho}{J^+} \qquad (۱۲-۴)$$

البته در رابطه‌ی فوق عمل تبخیر نیز لحاظ شده است و اگر عمل تبخیر از رابطه‌ی فوق حذف شود در این صورت رابطه‌ی زیر به دست می‌آید که فقط عمل تجدید فرومون سراسری را بازگو می‌کند:

$$\tau_{ij} \leftarrow \tau_{ij} + \frac{\rho}{(1 - \rho).J^+} \qquad (۱۳-۴)$$

دو مرحله‌ای کردن عمل فرومون‌ریزی نیز با هدف ایجاد تعادل میان جستجو و بهره‌برداری و یا به عبارت بهتر ایجاد تعادل بین گوناگونی و تاکید، انجام گرفته است.

۴-۴-۴ الگوریتم مورچه‌ی کمینه-بیشینه[۱] (MMAS)

یکی دیگر از نسخه‌های تغییر یافته‌ی الگوریتم مورچه‌ها، الگوریتم مورچه‌ی کمینه-بیشینه یا MMAS است که به وسیله *اشتوتزل*[۲] و *هووس*[۳] در سال ۱۹۹ [۳۵] معرفی گردید و در سال ۲۰۰۰ [۳۲] نوع دیگری از آن ارایه شد. تفاوت‌های عمده‌ی MMAS با الگوریتم AS عبارتند از [۱۹، ۳۲، ۳۵، ۳۹]:

---

۱ Min-Max Ant System

۲ Stützle

۳ Hoos

- ❖ فقط بهترین مورچه مجاز به فرومون‌ریزی بر روی مسیر حرکتش می‌باشد.
- ❖ مقدار فرومون بر روی همه یال‌ها بین دو مقدار τ_{min} و τ_{max} محدود شده است.
- ❖ در مرحله‌ی ابتدایی الگوریتم، مقدار فرومون همه یال‌ها برابر با τ_{max} قرار داده می‌شود.
- ❖ تجدید فرومون به صورت تناسبی انجام می‌گیرد. مقادیر بیشتر فرومون بر روی یال، کمتر از مقادیر کمتر فرومون، تقویت می‌شوند.
- ❖ می‌توان همه مقادیر فرومون‌ها را به صورت اختیاری برابر با τ_{max} قرار داد.

۴-۵ کاربردها

یکی از کاربردهای اولیه‌ی الگوریتم مورچه‌ها، حل مساله فروشنده دوره‌گرد است. در همه الگوریتم‌های مورچه، فرض بر این است که مورچه‌ها بر روی یک گراف در حال حرکت هستند. لذا استفاده از مساله فروشنده دوره‌گرد، برای به تصویر کشیدن قابلیت‌ها و ویژگی‌های الگوریتم‌های مورچه، یک انتخاب به طور کامل منطقی است. الگوریتم مورچه‌ها، برای مساله فروشنده دوره‌گرد جواب‌های مناسبی به دست آورده است. به خصوص اگر الگوریتم مورچه‌ها با یک روش بهینه‌سازی محلی دیگر ترکیب شود، نتایج خوبی به دست می‌دهد [۱۶، ۱۹، ۲۲، ۲۴، ۲۶-۲۸، ۳۰].

مساله دیگری که می‌توان با الگوریتم‌های مورچه آن را حل کرد، مساله مسیریابی خودرو (VRP[۱]) یا VPR می‌باشد که شباهت‌هایی به مساله TSP نیز دارد. در این مساله عده‌ای از مشتریان وجود دارند که می‌باید فقط یک بار خدمات رسانی شوند. خدمات از طریق تعدادی خودرو انجام می‌پذیرد که از یک انبار مشخص کار خود را شروع می‌کنند و در همان انبار نیز به کار خود پایان می‌دهند. هدف از حل مساله، کمینه کردن تعداد خودروها است، به نحوی که برخی قیود همچون بیشترین ظرفیت مجاز هر خودرو، بیشترین طول مسیر مجاز برای هر خودرو، بیشترین مقدار مجاز برای سوخت مصرفی به وسیله خودروها و بیشترین مهلت زمانی موجود برای انجام خدمات، برآورده شوند [۳۱، ۳۶، ۳۷].

یکی دیگر از مسایل قابل حل با استفاده از الگوریتم مورچه‌ها، *مساله* تخصیص درجه۲ (QAP[۲]) است که از مسایل مهم در زمینه‌ی بهینه‌سازی ترکیبی و تحقیق در عملیات[۳] است. این مساله از دسته‌ی مسایل *مکان‌یابی تاسیسات*[۴] است که کاربردهای فراوانی در مهندسی صنایع، مدیریت شهری، شهرسازی، مدیریت منابع و مدیریت محیط زیست دارند. در این مساله فرض بر این است که n واحد

---

[۱] Vehicle Routing Problem

[۲] Quadratic Assignment Problem

[۳] operations research

[۴] facility location

(خدماتی، تولیدی یا تاسیساتی) و مجموعه‌ای از n مکان وجود دارند. برای هر دو مکان فاصله‌ای تعریف شده است و برای هر دو واحد نیز یک *جریان* یا *وزن* انتقالی تعریف شده است. به عنوان مثال، مقدار مواد اولیه یا محصولات، که بین دو واحد جابه‌جا می‌شود، نمونه‌ای از جریان می‌باشد. منظور از حل مساله، تخصیص هر کدام از واحدها به یک مکان است به نحوی که مجموع حاصل‌ضرب‌های فواصل و جریان‌های متناظر کمینه گردد. به دلیل جملات ضربی که در تابع هزینه‌ی این مساله ظاهر می‌شوند، تابع هزینه و مساله با نام درجه دو شناخته می‌شوند [۱۹، ۳۸-۴۰].

یکی دیگر از کاربردهای مهم برای الگوریتم‌های مورچه حل مسایل مسیریابی در شبکه‌های مخابراتی است. هدف از حل چنین مساله‌ای انتقال هر چه سریع‌تر اطلاعات از طریق شبکه‌های مخابراتی با توجه به یک جدول مسیریابی است، به نحوی که کیفیت و کارآیی شبکه به بیشترین حد ممکن برسد. به عنوان مثال، می‌بایست خلوت‌ترین و در عین حال نزدیک‌ترین خطوط ارتباطی می‌بایست برای انتقال اطلاعات مورد استفاده قرار گیرند. به طور معمول این مساله بر روی شبکه‌های کامپیوتری و در سطوح مختلف، از شبکه‌های محلی تا شبکه جهانی، مطرح می‌شود. نمونه‌های تغییر یافته‌ای از این مساله، به حل مشکلات ازدحام در سطح شهر مربوط می‌شوند [۱۹،۱۶، ۲۲].

۴-۶ خلاصه فصل

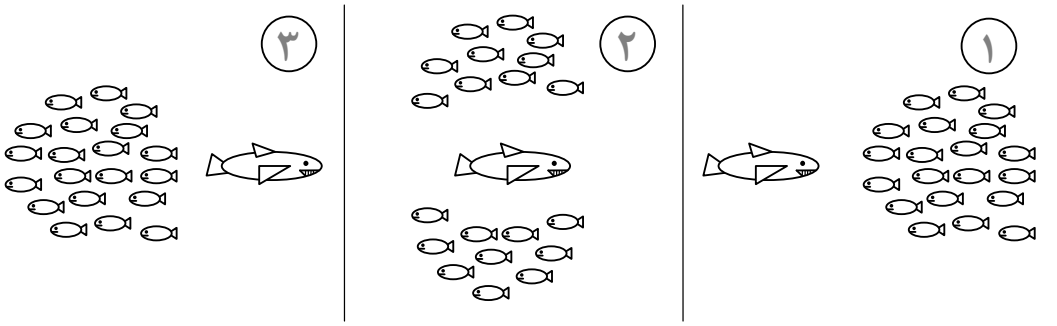
در این فصل در ابتدا الگوریتم بهینه‌سازی کلونی مورچه‌ها در دو بخش مختلف الگوریتم ساده شده مورچه‌ها و الگوریتم مورچه‌ها شرح داده شدند. سپس کاربرد این روش برای حل مساله فروشنده دوره‌گرد معرفی شد. پس از آن برخی از نسخه‌های تغییر یافته الگوریتم مورچه‌ها از قبیل مورچه‌ها و نخبه‌گرایی، مورچه‌ها و یادگیری تقویتی، سامانه کلونی مورچه‌ها و مورچه کمینه-بیشینه معرفی شدند. در نهایت کاربردهای مختلف این روش در مسایل مختلف مطرح شدند. در ادامه در فصل آتی الگوریتم بهینه‌سازی انبوه ذرات معرفی خواهد شد.

فصل پنجم

الگوریتم بهینه‌سازی انبوه ذرات[1]

۵-۱ مقدمه

برای برخی از حیوانات که به صورت گروهی زندگی می‌کنند از جمله دسته‌های ماهی، رفتارهای پیچیده‌ای به هنگام حرکت قابل مشاهده هستند. این در حالی است که هر کدام از اعضای جمع به اطلاعات محدودی دسترسی دارند و فقط از موقعیت عده‌ی اندکی از همسایگان‌شان خبر دارند. به عنوان مثال، یک دسته از ماهی‌ها، همان طور که در شکل ۵-۱ دیده می‌شود، می‌توانند خطر یک شکارچی را دفع کنند. در ابتدا گروه به دو قسمت تقسیم می‌شود و سپس از نو ساخته می‌شود. اما در هر حالتی، نزدیکی و فشردگی کل جمع، از طرف همه‌ی ماهی‌ها کنترل می‌شود [۱].



شکل ۵-۱: یک گروه از ماهی‌ها که خطر یک شکارچی را پشت سر می‌گذارد [۱].

[1] Particle Swarm Optimization

در چنین مجموعه‌ای، هر کدام از حیوانات فقط از چند قانون ساده تبعیت می‌کنند و رفتارهای پیچیده‌ای که در کل جمع قابل مشاهده هستند، چیزی جز ترکیب این قوانین ساده نیستند. هر کدام از ماهی‌ها در یک دسته، از موقعیت، جهت حرکت و سرعت ماهی‌های نزدیکش خبر دارد و با استفاده از این اطلاعات و پیروی از چند قانون ساده، خود را با جمع تطبیق می‌دهد [۱۶، ۱۹، ۲۳-۲۵]. به عنوان مثال:

- همواره سعی داشته باش که به طور نسبی، در نزدیکی سایرین حرکت کنی.
- سعی کن در جهتی که بقیه حرکت می‌کنند، حرکت بکنی.
- تقریبا با همان سرعتی که دیگران حرکت می‌کنند، حرکت کن.

از قوانین ساده‌ای هستند که هر ماهی در یک دسته رعایت می‌کند و حاصل تبعیت از چنین قوانینی، حرکت‌های پیچیده‌ای است که در مجموع به وسیله دسته‌ی ماهی‌ها انجام می‌گیرد.

این رفتارها و مجموعه رفتارهایی که در مورد حشرات و در مقدمه‌ی فصل ۴ توضیح داده شد، همگی با اصلی به نام *خود-ترتیبی*[۱] شناخته می‌شوند [۱۹]. این اصطلاح در علوم مختلف، همچون فیزیک و زیست‌شناسی، تعاریف جداگانه دارد. یک تعریف دقیق به این صورت است [۴۴]:

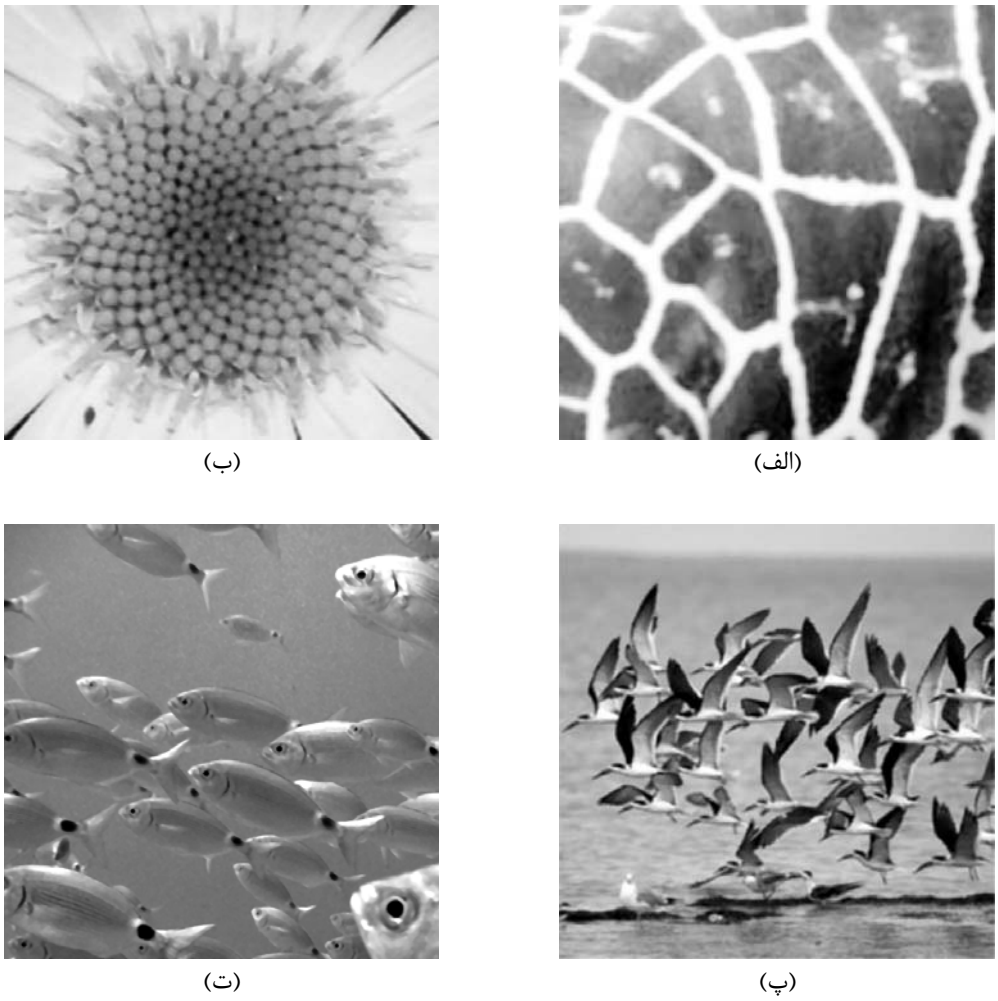
> **«*خود-ترتیبی* فرآیندی است که در آن الگوی کلی یک سیستم پیچیده، فقط در اثر تعداد زیادی از تعاملات مرتبه-پایین و درونی شکل می‌گیرد. علاوه بر این، قوانینی که بر تعامل بین عناصر تشکیل دهنده‌ی سیستم حاکم هستند فقط از اطلاعات محلی استفاده می‌کنند و هیچ وقت از اطلاعات سراسری بهره نمی‌برند.»**

در شکل ۵-۲، چهار نمونه از الگوهای موجود در طبیعت مشاهده می‌شود. نمونه (الف) طرح پوست یک زرافه را نشان می‌دهد. طرز چینش و تقسیم‌بندی در پوست زرافه، با چند قانون ساده قابل بیان است. نمونه (ب) نظم موجود در بخش مرکزی گل آفتاب‌گردان را نشان می‌دهد. مارپیچی که دانه‌های گل آفتاب‌ردان تشکیل می‌دهند، با یک سری *فیبوناچی*[۲] قابل توصیف است. نمونه‌های (پ) و (ت) نیز مربوط به عملکرد جمعی پرندگان و ماهی‌ها هستند که با وجود تمام پیچیدگی‌هایشان، از ترکیب و تکرار چند قانون ساده ساخته شده‌اند. در تمام این مثال‌ها، ترکیب و تکرار چند قانون ساده، باعث ایجاد یک نظم پیچیده شده است. البته تفاوتی که بین دو نمونه اول با دو نمونه دیگر وجود دارد در ساکن بودن سیستم کلی است. در نمونه‌های (پ) و (ت) قوانینی برای حرکت نیز وجود دارند، اما در دو نمونه اول، فقط قوانینی برای طرز چینش قابل توصیف هستند [۱۹].

---

[۱] Self-organization

[۲] fibonacci



شکل ۵-۲ : چند مثال از الگوهای موجود در طبیعت [۱].

نحوه‌ی ارتباط میان عناصر یک سیستم، نکات جالبی را در دل خود دارد. برای عملکرد جمعی، وجود داشتن یک جریان سیال اطلاعاتی، ضروری است. برای ایجاد اطلاعات بهتر از روی اطلاعات فعلی، می‌بایست بخش‌های مهم و مفید از میان اطلاعات موجود انتخاب و سپس تقویت شوند. این تقویت در سیستم‌های طبیعی به طور معمول به صورت یک *پس‌خورد*[۱] مثبت می‌باشد. به دلیل خواص ناپایدار کننده پس‌خورد مثبت، الگوهای طبیعی دارای پس‌خوردهای منفی نیز می‌باشند، تا به این وسیله سیستم از نظر پایداری دچار مشکل نشود [۱۶، ۱۹، ۲۳].

[۱] feedback

۵-۲ الگوریتم بهینه‌سازی انبوه ذرات (PSO)

جیمز کِندی[۱]، روانشناس اجتماعی، و *راسِل سی اِبِرهارت*[۲]، مهندس برق، صاحبان اصلی ایده‌ی الگوریتم PSO می‌باشند. آن‌ها در ابتدا قصد داشتند که با بهره‌گیری از مدل‌های اجتماعی و روابط موجود اجتماعی، نوعی از هوش محاسباتی را به وجود بیاورند که به توانایی‌های فردی ویژه نیازی نداشته باشد. اولین شبیه‌سازی آن‌ها که در سال ۱۹۹۵ [۴۱، ۴۲] انجام گردید، آن‌ها را به سمت شبیه‌سازی رفتار پرندگان برای یافتن دانه رهنمون کرد. این کار تحت تاثیر کار هِپنِر[۳] و گرِناندِر[۴] بود، که در سال ۱۹۹۰ برای شبیه‌سازی رفتار پرندگان به صورت یک سیستم غیر خطی انجام گرفته بود. کار کندی و ابرهارت، منجر به ایجاد الگوریتمی قوی برای بهینه‌سازی، به نام الگوریتم بهینه‌سازی گروه ذرات یا PSO شد [۱۶، ۱۹، ۵۵]. در الگوریتم PSO، تعدادی از موجودات وجود دارند، که به آن‌ها ذره گفته می‌شود و در فضای جستجوی تابعی که قصد کمینه کردن (و یا بهینه کردن) مقدار آن را داریم، پخش شده‌اند. هر ذره مقدار تابع هدف را در موقعیتی از فضا که در آن قرار گرفته است، محاسبه می‌کند. سپس با استفاده از ترکیب اطلاعات محل فعلی‌اش و بهترین محلی که در گذشته در آن بوده است و هم‌چنین اطلاعات یک یا چند ذره از بهترین ذرات موجود در جمع، جهتی را برای حرکت انتخاب می‌کند. همه‌ی ذرات جهتی برای حرکت انتخاب می‌کنند و پس از انجام حرکت، یک مرحله از الگوریتم به پایان می‌رسد. این مراحل چندین بار تکرار می‌شوند تا آن که جواب مورد نظر به دست بیاید. در واقع انبوه ذرات که مقدار کمینه‌ی یک تابع را جستجو می‌کنند، همانند دسته‌ای از پرندگان عمل می‌کنند که به دنبال غذا می‌گردند [۱۶، ۱۸، ۱۹، ۲۳، ۲۴، ۴۱، ۵۵].

هر ذره در الگوریتم PSO از سه بردار d بُعدی تشکیل شده است که d بُعد فضای جستجو می‌باشد. برای ذره‌ی iاُم این سه بردار عبارتند از: x^i موقعیت فعلی ذره، v^i سرعت حرکت ذره و $x^{i,best}$ بهترین موقعیتی که ذره تا به حال تجربه کرده است. x^i مجموعه‌ای از مختصات است که موقعیت فعلی ذره را نمایش می‌دهد. در هر مرحله‌ای که الگوریتم تکرار می‌شود، x^i به عنوان یک جواب برای مساله محاسبه می‌شود. اگر این موقعیت بهتر از جواب‌های پیشین باشد در $x^{i,best}$ ذخیره می‌شود. f^i مقدار تابع هدف در x^i و $f^{i,best}$ مقدار تابع هدف در $x^{i,best}$ است که هر دو از عناصر تشکیل دهنده‌ی هر ذره به حساب می‌آیند. ذخیره کردن مقدار $f^{i,best}$ برای انجام مقایسه‌های بعدی، ضروری است. اما ذخیره کردن مقدار f^i ضروری نمی‌باشد. در هر تکرار x^i و v^i جدیدی به دست می‌آیند و منظور از اجرای الگوریتم، بهتر کردن $x^{i,best}$ و به احتمال x^i است.

الگوریتم PSO چیزی فراتر از یک مجموعه‌ی ذرات است. هیچ‌کدام از ذرات قدرت حل هیچ مساله‌ای را ندارند، بلکه هنگامی می‌توان به حل مساله امیدوار شد که آن‌ها با هم‌دیگر ارتباط و تعامل داشته

---

[۱] James Kennedy
[۲] Russell C. Eberhart
[۳] Heppner
[۴] Grenander

باشند. در واقع برای انبوه ذرات، حل مساله، یک مفهوم اجتماعی است که از رفتار تک‌تک ذرات و تعامل میان آن‌ها به وجود می‌آید. بهترین موقعیتی که به وسیله همه‌ی ذرات پیدا شده است به صورت x^{gbest} نشان داده می‌شود که با مقایسه‌ی مقادیر $f^{i,best}$ به ازای همه‌ی ذرات و از میان $x^{i,best}$ها انتخاب می‌شود. مقدار تابع هدف در x^{gbest} به صورت f^{gbest} نشان داده می‌شود. اگر تعداد ذرات موجود در جمعیت، n باشد، آن‌گاه می‌توان روابط زیر را نوشت:

$$x^{i,best}[t] = \underset{\tau \le t}{\arg\min} f\left(x^{i}[\tau]\right) = \arg\min\left\{f\left(x^{i}[t]\right), f\left(x^{i,best}[t-1]\right)\right\} \qquad (۱-۵)$$

$$f^{i,best}[t] = f\left(x^{i,best}[t]\right) = \min_{\tau \le t} f^{i}[\tau] = \min\left\{f^{i}[t], f^{i,best}[t-1]\right\} \qquad (۲-۵)$$

$$x^{gbest}[t] = \underset{i=1,\ldots,n}{\arg\min} f\left(x^{i,best}[t]\right) \qquad (۳-۵)$$

$$f^{gbest}[t] = f\left(x^{gbest}[t]\right) = \min_{i=1,\ldots,n} f^{i,best}[t] \qquad (۴-۵)$$

در مرحله‌ی ابتدایی الگوریتم، ذرات با موقعیت‌ها و سرعت‌های تصادفی ایجاد می‌شوند. در طی اجرای الگوریتم، موقعیت و سرعت هر ذره در مرحله‌ی $t+1$اُم از الگوریتم، از روی اطلاعات مرحله‌ی قبلی ساخته می‌شوند. اگر z_j مولفه‌ی jاُم از بردار z باشد، آن‌گاه روابطی که سرعت و موقعیت ذرات را تغییر می‌دهند، عبارتند از:

$$v_j^i[t+1] = wv_j^i[t] + c_1 r_1\left(x_j^{i,best}[t] - x_j^i[t]\right) + c_2 r_2\left(x_j^{gbest}[t] - x_j^i[t]\right) \qquad (۵-۵)$$

$$x_j^i[t+1] = x_j^i[t] + v_j^i[t+1] \qquad (۶-۵)$$

در این روابط، w ضریب اینرسی، r_1 و r_2 اعدادی تصادفی در بازه‌ی $[0,1]$ با توزیع یکنواخت و هم‌چنین c_1 و c_2 ضرایب یادگیری هستند. r_1 و r_2 باعث می‌شوند که نوعی گوناگونی در جواب‌ها به وجود بیاید و به این نحو جستجوی کاملی روی فضا انجام پذیرد. c_1 ضریب یادگیری مربوط به تجارب شخصی هر ذره است و در مقابل c_2 ضریب یادگیری مربوط به تجارب کل جمع می‌باشد. از معادله‌ی (۵-۵) می‌توان به این نتیجه رسید که، هر ذره به هنگام حرکت، (الف) جهت حرکت قبلی خود، (ب) بهترین موقعیتی را که در آن قرار داشته است و (پ) بهترین موقعیتی را که به وسیله کل جمع تجربه شده است، در نظر می‌گیرد. در برخی موارد، روابط (۵-۵) و (۵-۶) به صورت زیر، برای همه‌ی ابعاد جمع‌بندی می‌شود:

$$v^i[t+1] = wv^i[t] + c_1 R_1 \otimes \left(x^{i,best}[t] - x^i[t]\right) + c_2 R_2 \otimes \left(x^{gbest}[t] - x^i[t]\right) \qquad (۷-۵)$$

$$x^i[t+1] = x^i[t] + v^i[t+1] \qquad (۸-۵)$$

که در آن، R_1 و R_2 دو بردار هم اندازه با بعد فضای جستجو هستند که مولفه‌هایشان اعداد تصادفی مستقل با توزیع یکنواخت و در بازه‌ی $[0, 1]$ هستند. هم‌چنین علامت $\otimes$ نشان دهنده‌ی عمل ضرب عضو به عضو برای ماتریس‌ها است. به منظور محدود کردن میزان حرکت هر ذره، مقدار مولفه‌های سرعت ذرات در بازه‌ی $[-v_{max}, v_{max}]$ در نظر گرفته می‌شود و مقادیر بزرگ‌تر یا کوچک‌تر نیز به این بازه تصویر می‌شوند. البته فرض بر این است که عرض فضای جستجو در تمام ابعاد ثابت و برابر با s باشد. در این صورت به طور معمول، $v_{max} = ps$ در نظر گرفته می‌شود که $p \in [0.1, 1]$ است [۲۳، ۲۴]. در شکل ۵-۳ مراحل الگوریتم PSO که در بالا توضیح داده شد، آمده است.

الگوریتم بهینه‌سازی انبوه ذرات

- n ذره بساز.
- برای تمام ذرات، سرعت و موقعیتی تصادفی ایجاد کن.
- **تا زمانی که** شرایط خاتمه محقق نشده‌اند:
  - یک واحد به t اضافه کن.
  - مقدار تابع هدف را به ازای هر ذره محاسبه کن.
  - **به ازای** i از یک تا n:
    - $x^{i,best}[t]$ را محاسبه کن.
  - **مقدار بعدی** i.
  - $x^{gbest}[t]$ را محاسبه کن .
  - **به ازای** i از یک تا n:
    - **به ازای** j از یک تا d:

$$v_j^i[t+1] = wv_j^i[t] + c_1 r_1 \left(x_j^{i,best}[t] - x_j^i[t]\right) + c_2 r_2 \left(x_j^{gbest}[t] - x_j^i[t]\right)$$

$$x_j^i[t+1] = x_j^i[t] + v_j^i[t+1]$$

    - **مقدار بعدی** j .
  - **مقدار بعدی** i.

شکل ۵-۳ : مراحل الگوریتم PSO [۱].

۵-۳ پارامترهای PSO

ضریب اینرسی w بر روی هم‌گرایی الگوریتم PSO تاثیر مستقیم دارد. در واقع می‌توان به واسطه‌ی ضریب اینرسی، تاثیر سرعت‌های گذشته را بر سرعت‌های زمان حال کنترل نمود. می‌توان برای برقراری موازنه‌ی بهتر میان *جستجوی سراسری*[۱] و *جستجوی محلی*[۲] مقدار w را تغییر داد. مقدار زیاد برای w باعث می‌شود که ذرات موجود در الگوریتم، به جستجوی مناطق جدیدتر روی بیاورند و یک جستجوی سراسری را انجام دهند. در مقابل یک مقدار کم برای w باعث می‌شود که ذرات در منطقه‌ی محدودی بمانند و در واقع یک جستجوی محلی را انجام دهند. جستجوی محلی برای دقیق‌تر کردن جواب‌های فعلی مناسب است و جستجوی سراسری برای یافتن جواب‌های بهتری که به احتمال در جاهای ناشناخته از فضای جستجو وجود دارند، به کار می‌رود [۲۳، ۲۴].

معادله‌ی (۵-۷) به صورت زیر قابل باز نویسی است:

$$v^i[t+1] = wv^i[t] + F^i[t] \qquad (۹-۵)$$

که در آن $F^i[t]$، به معنی یک نیروی خارجی است و به صورت زیر تعریف شده است:

$$F^i[t] = c_1R_1 \otimes \left(x^{i,best}[t] - x^i[t]\right) + c_2R_2 \otimes \left(x^{gbest}[t] - x^i[t]\right) \qquad (۱۰-۵)$$

تغییرات سرعت ذره، یا شتابی که به ذره وارد می‌شود، به این صورت قابل محاسبه است:

$$\Delta v^i[t+1] = v^i[t+1] - v^i[t] = F^i[t] - (1-w)v^i[t] \qquad (۱۱-۵)$$

در واقع ضریب $1-w$ به عنوان یک ضریب اصطکاک عمل می‌کند و می‌توان w را به صورت ضریب *سیالی*[۳] محیطی که ذره در آن در حال حرکت است، در نظر گرفت. اگر معادله‌ی (۵-۹) را به صورت معادله‌ی فضای حالت یک سیستم گسسته فرض شود، مقادیر بزرگ‌تر از یک برای w، باعث ناپایدار شدن سیستم ذرات می‌شود. از طرفی به ازای مقادیر کمتر برای w، سرعت هم‌گرایی سامانه بیشتر خواهد شد [۵۵].

یک مقدار مناسب برای w، باعث ایجاد تعادل بین جستجوی‌های محلی و سراسری می‌شود و در اغلب اوقات باعث کاهش تعداد تکرارهای لازم برای هم‌گرایی به یک جواب مناسب، می‌شود. در الگوریتم ابتدایی PSO مقدار w ثابت در نظر گرفته می‌شد. اما نتایج عملی حاکی از آن هستند که بهتر است مقدار w در مراحل ابتدایی، یک مقدار بزرگ در نظر گرفته شود تا یک جستجوی کامل و سراسری از

---

۱ global exploration (global search)

۲ local exploration (local search)

۳ fluidity

فضای جستجو صورت گیرد. سپس در طی مراحل اجرای الگوریتم، مقدار w به تدریج کاهش داده می‌شود تا الگوریتم به مرز هم‌گرایی نزدیک شود و جواب‌های دقیق‌تری به دست بدهد. با در نظر گرفتن w به صورت ضریب سیالی محیط، مقدار بیشتر برای w، به معنی راحت‌تر بودن حرکت در محیط است و محیط دارای *گران‌روی*[۱] پایین‌تری است. با کمتر شدن مقدار w، حرکت ذرات در محیط سخت‌تر می‌شود و به این ترتیب ذرات با گران‌روی بیشتری مواجه می‌شوند. در این حالت امکان هم‌گرایی ذرات به سمت نقاط بهینه بیشتر می‌شود. به عنوان مثال در نظر گرفتن مقدار $1/2$ برای w و کاهش دادن تدریجی آن تا مقدار صفر، شیوه خوبی برای بیشتر مسایل است [۱۶، ۲۳، ۲۴، ۴۳، ۵۵].

هم‌چنین انتخاب w به صورت یک عدد تصادفی با توزیع یکنواخت در بازه‌ی $[0.5, 1]$ نتایج خوبی را به همراه داشته است [۴۵]. راه‌حل بهتر و البته پیچیده‌تر، استفاده از *روش‌های تطبیقی*[۲]، همانند *کنترل فازی*[۳] است [۴۶، ۴۷]. در این روش‌ها، با توجه به شرایط مساله، مقدار پارامترها به صورتی تطبیقی تعیین می‌شوند. عملکرد ضریب اینرسی در الگوریتم PSO همانند عملکرد دما در روش *شبیه‌ساز* سرد کردن فلزات (SA[۴]) است که در فصل سوم توضیح داده شده است. دماهای بالاتر در الگوریتم SA، باعث آزادی عمل بیشتر برای اتم‌ها یا جواب‌های مساله هستند و به تدریج که از دما کاسته می‌شود، تغییرات کمتری از طرف الگوریتم پذیرفته می‌شود.

با وجود آن که در نسخه‌های اولیه‌ی الگوریتم PSO، عامل میرا کننده‌ای همچون محدود کردن سرعت در بازه‌ی $[-v_{max}, v_{max}]$ در نظر گرفته شده بود اما به ضرورت نیاز به چنین عاملی توجه نشده بود. اگر الگوریتم بدون در نظر گرفتن محدودیت‌های سرعت اجرا شود، در عرض چندین تکرار، سرعت ذرات به شدت افزایش می‌یابد و به مقادیر غیر قابل قبول می‌رسد. کِندی ضمن تحقیقات خود، در سال ۱۹۹۸، دریافت که برای ذرات تک-بُعدی که به صورت غیر تصادفی حرکت می‌کنند، اگر مقدار $c_1 + c_2$ بین صفر و ۴ باشد، مسیرهایی که ذرات طی می‌کنند قابل قبول‌تر می‌باشند. با تحلیل‌هایی که بر روی سیستم حرکت ذرات انجام شد، راهبردی برای تعیین ضرایب یادگیری c_1 و c_2 ایجاد شده است که (الف) از ناپایدار شدن سیستم حرکتی ذرات جلوگیری می‌کند، (ب) هم‌گرایی ذرات را تضمین می‌کند و (پ) نیازی به تعریف پارامتر v_{max} وجود ندارد. هم‌چنین به واسطه‌ی تحلیل‌های انجام شده، روشی برای تعیین مقادیر حدسی برای ضرایب یادگیری نیز ارایه شده است [۴۵، ۴۸، ۵۵].

کلِرک[۵] و کِندی در تحقیقات‌شان به این نتیجه رسیدند که راه‌های زیادی برای تعیین مقادیر ضرایب یادگیری وجود دارد. یکی از ساده‌ترین روش‌ها برای تعیین مقادیر مناسب برای ضرایب یادگیری به این ترتیب است [۴۵، ۴۸، ۵۵].

---

[۱] viscosity

[۲] adaptive methods

[۳] fuzzy control

[۴] Simulated Annealing

[۵] Clerc

$$\begin{aligned} w &= \chi \\ c_1 &= \chi\phi_1 \\ c_2 &= \chi\phi_2 \end{aligned} \qquad (۱۲-۵)$$

که در آن، ϕ_1 و ϕ_2 اعدادی مثبت هستند و به نحوی انتخاب می‌شوند که $\phi = \phi_1 + \phi_2 \geq 4$ باشد. χ نیز از روی رابطه زیر تعیین می‌شود:

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \qquad (۱۳-۵)$$

کِلِرک مقادیر $\phi_1 = \phi_2 = 2.05$ را پیشنهاد می‌کند. به این ترتیب مقادیر پارامترهای الگوریتم عبارتند از: $w \simeq 0.7298$، $c_1 = c_2 \simeq 1.4962$ [۲۴، ۴۵، ۵۵]. با استفاده از روابط فوق، ذرات بدون نیاز به کمیت محدود کننده v_{max} هم‌گرا می‌شوند. با این همه، تحقیق‌ها و آزمایش‌های اِبِرهارت و شی[۱] به این نتیجه رسیده است که نتایج بهتر با در نظر گرفتن $p = 1$ در رابطه $v_{max} = ps$ به دست می‌آید [۲۴، ۵۵]. یعنی، حد بیشینه‌ی سرعت v_{max}، برابر با عرض فضای جستجو در نظر گرفته شود. و این همان الگوریتم معیاری است که امروزه به نام PSO شناخته می‌شود. در این الگوریتم، به طور معمول $c_1 = c_2 = 2$ در نظر گرفته می‌شود [۱۶، ۲۳، ۴۱].

الگوریتم PSO را می‌توان به شکل مجموعه‌ای از بردارها تصور کرد که، در فضایی متشکل از تجارب خصوصی هر ذره و برخی از ذرات دیگر، نوسان می‌کند. در حالت کلی، هر ذره با عده‌ای دیگر از ذرات دارای ارتباط است که به طور معمول این ارتباط دو طرفه می‌باشد و به نام رابطه همسایگی یا مجاورت شناخته می‌شود. مجموعه‌ی ذراتی که با یک ذره دارای ارتباط همسایگی هستند، به نام مجموعه‌ی همسایگی شناخته می‌شوند. برای ذره‌ی iاُم، بهترین موقعیتی که به وسیله همسایگانش تجربه شده است، به صورت $x^{i,nbest}$ نمایش داده می‌شود. $x^{i,nbest}$ یکی دیگر از مواردی است که در تصمیم‌گیری هر ذره تاثیر دارد. البته باید توجه کرد که مجموعه‌ی همسایه‌های یک ذره، شامل خود آن ذره نیز می‌شود. یعنی یک ذره، همسایه‌ی خودش نیز می‌باشد. کِنِدی و اِبِرهارت، دو الگوی مختلف برای PSO پیشنهاد دادند. *الگوی بهینه محلی*[۲] و *الگوی بهینه سراسری*[۳] [۲۳، ۲۴].

الگوی بهینه سراسری، چیزی است که تاکنون در این نوشتار به بررسی آن پرداخته شده است. اگر در روابط (۵-۵) و (۵-۷) به جای x^{gbest} از $x^{i,nbest}$ استفاده شود، به الگوی بهینه محلی رسیده می‌شود. در الگوی بهینه سراسری، یک ذره در جمع وجود دارد که به عنوان جاذب است و سایر ذرات را به سمت خود جذب می‌کند و به احتمال همه‌ی ذرات در محل بهترین ذره، هم‌گرا شوند. اما در الگوی بهینه محلی، چندین جاذب در جمع وجود دارند و هر ذره فقط به سمت بهترین همسایه‌ی خود جذب می‌شود. اگر دامنه‌ی تعریف همسایگی به همه‌ی جمع توسعه یابد، آنگاه الگوی بهینه محلی با الگوی

---

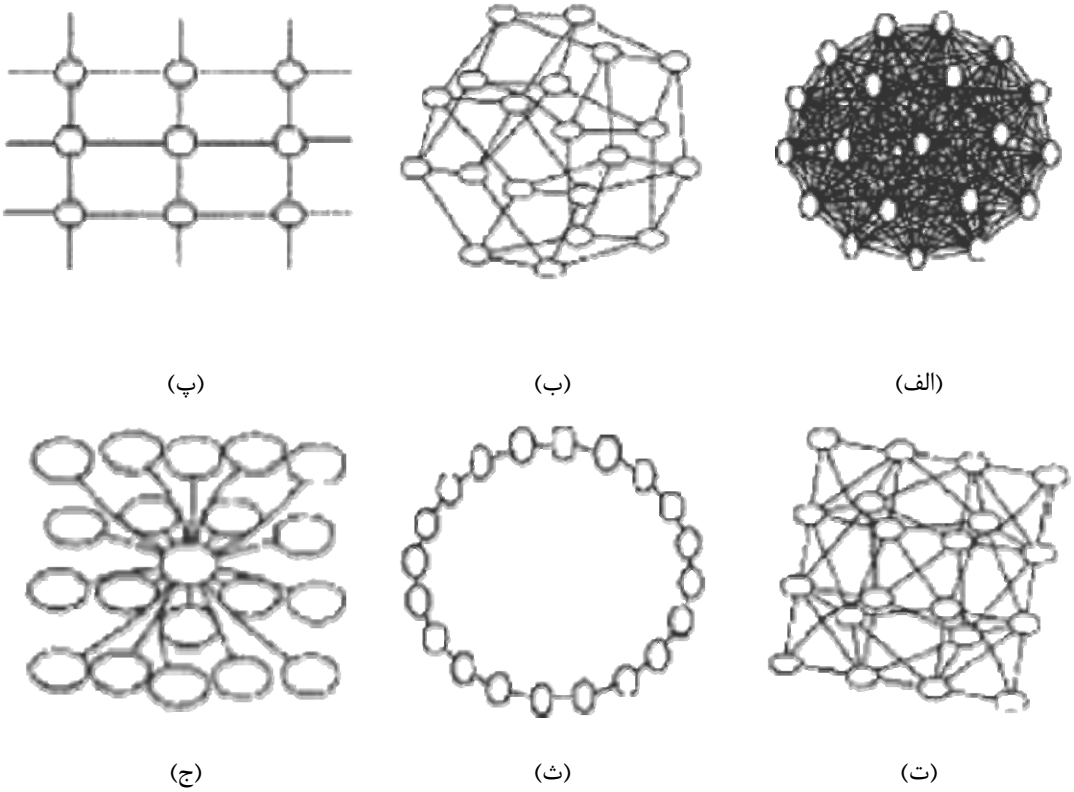
[۱] Shi

[۲] local best model

[۳] global best model

بهینه سراسری معادل خواهد بود. برای مسایل ساده که جواب یگانه‌ای دارند، الگوی بهینه سراسری همیشه توصیه می‌شود. هم‌چنین برای توابع پیچیده، الگوی بهینه محلی با تعریف همسایگی متغیر مناسب می‌باشد [۲۴، ۴۹-۵۱، ۵۵].

کِندی و مِندِس[۱] همسایگی‌های مختلف را، که می‌توان در میان ذرات تعریف نمود، مورد مطالعه قرار دادند. در حالت کلی، می‌توان روابط همسایگی را به دو نوع اصلی تقسیم کرد: (الف) همسایگی مبتنی بر فاصله و (ب) همسایگی مبتنی بر ارتباط‌های اجتماعی. در همسایگی مبتنی بر فاصله، به طور معمول فاصله‌ی هندسی یا اقلیدسی دو ذره در فضای جستجو، مشخص می‌کند که آیا دو ذره با هم همسایه هستند یا نه. در همسایگی مبتنی بر روابط اجتماعی، همسایگی فقط به معنی نزدیکی و یا دوری فیزیکی نمی‌باشد. در این نوع همسایگی، الگوی خاصی، که به احتمال متغیر نیز باشد، مشخص می‌کند که آیا دو ذره با هم همسایه هستند یا نه. اشکال مختلفی از همسایگی‌ها در شکل ۵-۴ آمده است [۲۴، ۴۹]:



شکل ۵-۴ : اشکال مختلف همسایگی [۱].

[۱] mendes

رابطه همسایگی در شکل ۵-۴ به صورت گراف‌هایی نمایش داده شده است. به این معنی که اگر دو ذره با هم همسایه باشند، یالی از گراف آن دو را به هم وصل می‌کند. اگر همسایگی به صورت سراسری تعریف شود، آن‌گاه گراف همسایگی به یک *گراف کامل*[۱] تبدیل می‌شود که همه‌ی رأس‌های آن به هم‌دیگر متصل هستند. در شکل ۵-۴ (الف) یک گراف کامل نشان داده شده است. همسایگی *فون نیومن*[۲]، نوع خاصی از همسایگی است که در آن هر ذره با یک ذره در هر کدام از جهت‌های بالا، پایین، چپ و راست همسایه است. شکل ۵-۴ (ب) گراف متناظر با تعریف همسایگی فون نیومن را، در حالت سه بُعدی، نشان می‌دهد. شکل ۵-۴ (پ) همان همسایگی است که مسطح شده و به صورت دو بُعدی در آمده است. در شکل ۵-۴ (ت) نوع دیگری از همسایگی موسوم به همسایگی هرمی نشان داده شده است. یکی از معمول‌ترین تعاریف مورد استفاده برای همسایگی، تعریف همسایگی حلقوی است که در شکل ۵-۴ (ث) نشان داده شده است. در شکل ۵-۴ (ج) همسایگی ستاره‌ای نشان داده شده است که در آن همه‌ی ذرات به ذره‌ای به نام رهبر متصل هستند که اطلاعات سایرین به وسیله او پردازش می‌شود.

مِندِس نحوه‌ی ارتباط ذرات با همسایگان‌شان را تغییر داد و روابط جدیدی را برای حرکت ذرات در فضای جستجو به وجود آورد. *الگوریتم بهینه‌سازی گروه ذرات با* اطلاعات کامل (FIPS[۳]) الگوریتمی است که به وسیله مِندِس ایجاد گردید [۴۹-۵۱]. در این الگوریتم، هر ذره از اطلاعات همه‌ی همسایه‌هایش برای تعیین جهت حرکت بهره می‌گیرد. روابط مربوط به FIPS عبارتند از:

$$v^i[t+1] = w v^i[t] + \frac{c}{n_i} \sum_{j \in \mathcal{N}_i} R_j \otimes \left(x^{j,best}[t] - x^i[t] \right) \qquad (۱۴-۵)$$

$$x^i[t+1] = x^i[t] + v^i[t+1] \qquad (۱۵-۵)$$

که در آن $\mathcal{N}_i$ مجموعه‌ی همسایه‌ها و $n_i = |\mathcal{N}_i|$ تعداد همسایه‌های ذره‌ی iاُم هستند. بردار R_j، برداری است که مولفه‌های آن، اعداد تصادفی با توزیع یکنواخت در بازه‌ی $[0, 1]$ هستند. c و w نیز ضرایبی ثابت و مشابه با ضرایب موجود در PSO معیار (استاندارد) هستند. اگر غیر از خود ذره، فقط اطلاعات بهترین همسایه در روابط بالا به کار گرفته شود، آن گاه FIPS به PSO تبدیل می‌شود. توجه کنید که در این روابط، فرض بر این است که هر ذره، همسایه‌ی خودش نیز می‌باشد. با پارامترهای مناسب، FIPS جواب‌های بهتری در تعداد تکرار کمتر نسبت به PSO به دست می‌آورد. اما عملکردش به شدت به نوع تعریف همسایگی وابسته است [۲۴، ۵۵].

---

[۱] complete graph

[۲] Von Neumann

[۳] Fully Informed Particle Swarm Optimization

۵-۴ برخی از نسخه‌های تغییر یافته‌ی PSO

طی دهه‌ی گذشته، تغییرات بسیار زیادی بر روی الگوریتم PSO اعمال شده است. عده‌ای از این تغییرات منجر به بهتر شدن کلی عملکرد الگوریتم PSO شده‌اند. عده‌ای از این تغییرات نیز، منجر به بهبود عملکرد الگوریتم برای رده‌ی خاصی از مسایل شده‌اند. در این بخش به تعدادی از مهم‌ترین تغییرات اعمال شده بر روی PSO اشاره شده است.

۵-۴-۱ الگوریتم PSO دودویی

در این نسخه از PSO که به وسیله کِنِدی و اِبرهارت در سال ۱۹۹۷ معرفی گردید، تغییری کوچک بر روی الگوریتم PSO اولیه انجام گرفته است، تا بتوان برای بهینه کردن کمیت‌های گسسته نیز از آن استفاده کرد [۱۶، ۱۴، ۵۲، ۵۳، ۵۵]. در این الگوریتم، سرعت به عنوان یک مقدار آستانه‌ی احتمالی استفاده شده است که معین می‌کند که مولفه‌ای از بردار موقعیت صفر یا یک باشد. فرض کنید x_j^i مقدار بیت jاُم از بردار دودویی نشان‌دهنده‌ی موقعیت ذره‌ی iاُم باشد. در این صورت رابطه زیر توصیف کننده نحوه‌ی عملکرد الگوریتم PSO دودویی است:

$$x_j^i[t] = \begin{cases} 1 & , \sigma < s\left(v_j^i[t]\right) \\ 0 & , \text{otherwise} \end{cases} \qquad (۱۶-۵)$$

که در آن σ عددی تصادفی، با توزیع یکنواخت و در بازه‌ی $[0, 1]$ است. $s(.)$ نیز تابع سیگموید[۱] است که به این صورت تعریف می‌شود:

$$s(z) \triangleq \frac{1}{1+e^{-z}} \qquad (۱۷-۵)$$

در PSO دودویی، سرعت ذرات، همانند الگوریتم PSO استاندارد تغییر داده می‌شود. کِنِدی و *اسپیرز*[۲] در سال ۱۹۹۸ این الگوریتم را با انواع مختلفی از الگوریتم‌های ژنتیک دودویی مقایسه نمودند. مسایلی که در این آزمایش‌ها حل می‌شدند، با روش اسپیرز به وجود آمده بودند. اسپیرز صاحب روشی برای طرح مسایل مختلف، با درجه‌ی سختی مورد نظر می‌باشد. در این آزمایش‌ها، PSO تنها الگوریتمی بود که توانست تمامی مسایل را، در یک بار تلاش، حل کند. هم‌چنین معلوم شد که الگوریتم PSO سریع‌تر از الگوریتم‌های ژنتیک با عمل‌گرهای تقاطع، جهش و یا هر دو است. فقط در مسایلی که بسیار ساده بودند، الگوریتم ژنتیک که فقط حاوی عمل‌گر جهش بود، سریع‌تر از PSO جواب را پیدا کرد.

---

[۱] Sigmoid

[۲] spears

در حالت کلی، به جای رابطه (۵-۱۶)، می‌توان رابطه زیر را برای تعریف عملکرد الگوریتم PSO دودویی به کار برد:

$$x_j^i[t+1] = \begin{cases} 1 & , \sigma < \pi\left(x_j^i[t], v_j^i[t+1], x_j^{i,best}[t], x_j^{gbest}[t]\right) \\ 0 & , \text{otherwise} \end{cases} \qquad (۱۸-۵)$$

این رابطه را می‌توان به این صورت نیز نوشت:

$$\Pr\left\{x_j^i[t+1]=1\right\} = \pi\left(x_j^i[t], v_j^i[t+1], x_j^{i,best}[t], x_j^{gbest}[t]\right) \qquad (۱۹-۵)$$

در این روابط، π تابعی است که می‌بایست مقدار آن بین صفر و یک باشد. تابع سیگموید که در رابطه (۵-۱۷) تعریف شده است، یک انتخاب بسیار ساده برای تابع π می‌باشد.

۵-۴-۲ الگوریتم PSO فازی یا FPSO

در الگوریتم PSO فازی شده، که برای حل مسایل بهینه‌سازی گسسته طراحی شده است، بردارهای موقعیت به صورت ماتریس‌هایی در نظر گرفته می‌شوند که عناصر آن‌ها کمیت‌های فازی هستند [۲۴، ۵۴]. در این بخش، برای توصیف نحوه‌ی عملکرد FPSO، چگونگی استفاده از آن را برای حل مساله معروف تخصیص کار[۱]، توضیح داده شده است [۲۴].

مساله تخصیص کار: فرض کنید که $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ مجموعه‌ای از کارها باشد که می‌بایست به وسیله مجموعه‌ای از دستگاه‌ها به صورت $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$ انجام شوند. هزینه‌ی انجام کار J بر روی دستگاه M، به صورت $C(J, M)$ در نظر گرفته می‌شود. همه‌ی کارهای مجموعه‌ی $\mathcal{J}$ می‌بایست انجام شوند، به نحوی که هزینه‌ی کلی انجام کارها، کمترین مقدار ممکن باشد. اگر Ψ یک راه حل برای مساله فوق باشد، می‌توان نوشت:

$$\Psi = \left\langle \left(J_1, M_{J_1}\right), \left(J_2, M_{J_2}\right), \ldots, \left(J_n, M_{J_n}\right) \right\rangle \qquad (۲۰-۵)$$

که در آن M_{J_i} دستگاهی است که می‌بایست کار J_i را انجام بدهد. هزینه‌ی متناظر با Ψ به صورت زیر قابل تعریف است:

$$C(\Psi) = \sum_{i=1}^{n} C(J_i, M_{J_i}) \qquad (۲۱-۵)$$

در الگوریتم FPSO، ارتباط بین کارها و دستگاه‌ها به صورت زیر در نظرگرفته می‌شود:

---

[۱] job scheduling

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}_{m \times n} \qquad (۲۲-۵)$$

که در آن x_{ij} یک کمیت فازی است و میزان تعلق J_j به مجموعه‌ی وظایف M_i را نشان می‌دهد. به عبارت بهتر، اگر $\mathcal{J}(M_i)$ مجموعه‌ی فازی وظایف دستگاه M_i باشد، آن‌گاه داریم:

$$x_{ij} = \mu_{\mathcal{J}(M_i)}(J_j) \qquad (۲۳-۵)$$

که در آن، $\mu_{\mathcal{J}(M_i)}$ تابع عضویت مجموعه‌ی $\mathcal{J}(M_i)$ می‌باشد. لذا مقادیر درایه‌های ماتریس X می‌بایست در بازه‌ی $[0, 1]$ باشد. از یک دیدگاه دیگر، x_{ij} احتمال آن است که دستگاه M_i کار J_j را انجام دهد. لذا می‌بایست شرط زیر برای ماتریس X برقرار باشد:

$$\sum_{i=1}^{m} x_{ij} = 1 \quad , \; j = 1, 2, \ldots, n \qquad (۲۴-۵)$$

و یا به عبارت بهتر، می‌بایست حاصل جمع هر کدام از ستون‌های ماتریس X، برابر با یک باشد. به طریق مشابه، ماتریس سرعت نیز قابل تعریف است:

$$V = \begin{bmatrix} v_{11} & v_{12} & \cdots & v_{1n} \\ v_{21} & v_{22} & \cdots & v_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ v_{m1} & v_{m2} & \cdots & v_{mn} \end{bmatrix}_{m \times n} \qquad (۲۵-۵)$$

معادلات (۵-۷) و (۵-۸) برای این الگوریتم، به صورت زیر بازنویسی می‌شوند:

$$V^{i}[t+1] = wV^{i}[t] + c_1 R_1 \otimes \left(X^{i,best}[t] - X^{i}[t]\right) + c_2 R_2 \otimes \left(X^{gbest}[t] - X^{i}[t]\right) \qquad (۲۶-۵)$$

$$X^{i}[t+1] = X^{i}[t] + V^{i}[t+1] \qquad (۲۷-۵)$$

که در آن، R_1 و R_1 ماتریس‌های از مرتبه‌ی $m \times n$ هستند که داریه‌هایشان اعداد تصادفی با توزیع یکنواخت و در بازه‌ی $[0, 1]$ می‌باشند. باقی کمیت‌های موجود در روبط، تغییر خاصی نکرده‌اند. با توجه به قیودی که بر روی مقادیر ماتریس X وجود دارند، می‌بایست ترتیبی اتخاذ کرد که مقادیر غیر مجاز برای ماتریس X اصلاح شوند. اول این که همواره می‌بایست مقادیر درایه‌های ماتریس X مثبت باشند. برای این منظور عناصری از ماتریس X که منفی باشند، صفر در نظر گرفته می‌شوند. از طرفی، می‌بایست جمع مقادیر هر ستون از ماتریس X، دقیقا یک باشد. به این منظور، هر کدام از درایه‌های ماتریس X را به مجموع درایه‌های واقع بر ستونش تقسیم می‌کنیم. نتیجه‌ی این عملیات، ماتریسی

مانند X^N می‌باشد که حرف N برای تاکید بر *هنجاری*[۱] یا نرمال بودن این ماتریس آمده است. درایه‌های X^N به این صورت قابل محاسبه هستند:

$$x_{ij}^N = \frac{b(x_{ij})}{\sum_{k=1}^{m} b(x_{kj})} \qquad (۲۸-۵)$$

که در آن، $b(.)$ تابعی است که به صورت زیر تعریف شده است:

$$b(x) \triangleq \frac{x + |x|}{2} = \begin{cases} x & , x \geq 0 \\ 0 & , x < 0 \end{cases} \qquad (۲۹-۵)$$

رابطه (۵-۲۸) را می‌توان به صورت خلاصه‌تری نیز نوشت:

$$x_{ij}^N = \frac{x_{ij} + |x_{ij}|}{\sum_{k=1}^{m} x_{kj} + |x_{kj}|} \qquad (۳۰-۵)$$

به این ترتیب می‌توان همه‌ی جواب‌هایی را که به دست می‌آیند، به شکلی مجاز تبدیل کرد.

با توجه به ماتریس X^N، می‌بایست چگونگی تخصیص کارها به دستگاه‌های مختلف مشخص شود. برای این کار، می‌بایست اطلاعاتی که را که در درون X^N به صورتی رمز شده وجود دارند، استخراج شوند. برای هر ستون از ماتریس، دستگاه متناظر با بیشترین مقدار انتخاب می‌شود. دستگاهی که انتخاب می‌شود، در مجموعه‌ی دستگاه‌های مشغول به کار وارد می‌شود. هرگاه دستگاهی انتخاب می‌شود، سطر متناظر با آن دستگاه، از بین سطرهای ماتریس حذف می‌شود. سپس به همین ترتیب برای ستون‌های بعدی نیز کار مشابهی انجام می‌گیرد تا جایی که همه‌ی کارها به دستگاه‌ها تخصیص داده شوند. در صورتی که تعداد دستگاه‌ها کمتر از تعداد کارها باشد، بعد از آن که برای همه‌ی دستگاه‌ها کاری انتخاب شد، دستگاه‌ها اجازه می‌یابند که کار دوم را به عهده بگیرند. با این ترتیب، جواب متناظر با ماتریس X^N به دست می‌آید [۲۴].

پس از آن که جواب‌های متناظر با تمام ذرات به دست آمدند، هزینه‌ی متناظر با هر جواب محاسبه می‌شود و بر اساس آن مقایسه بین ذرات انجام می‌گیرد. در نهایت با استفاده از روابط (۵-۲۶) و (۵-۲۷) ذرات در فضای جستجو حرکت می‌کنند و جواب‌های مناسب‌تری را به دست می‌آورند. همان طور که قبلا گفته شد، الگوریتم FPSO برای حل مسایل گسسته مناسب است و می‌توان با اعمال تغییرات مناسب بر الگوریتم ذکر شده در بالا، از آن برای حل انواع مسایل بهینه‌سازی گسسته بهره برد [۲۴، ۵۴].

---

[۱] normalized

۵-۵ کاربردها

الگوریتم PSO در طی سالیان گذشته، برای حل انواع مختلفی از مسایل مورد استفاده قرار گرفته است. الگوریتم PSO در حل مسایل پیچیده که چندین جواب بهینه محلی دارند مناسب است. برای حل چنین مسایلی یا روش دیگری وجود ندارد و یا در صورت وجود داشتن، جواب مناسبی به دست نمی‌آید [۵۵].

کاربردهایی که تاکنون برای PSO مطرح شده‌اند، در ۲۶ گروه مجزا تقسیم شده‌اند. البته برخی از کاربردها، به بیش از یک گروه متعلق می‌باشند. آماری که در این بخش بیان می‌شود، بر اساس مقالات موجود در پایگاه داده‌ی IEEE Xplore، تا اواخر سال ۲۰۰۷ می‌باشد. در حدود ۱۱۰۰ مقاله در مورد PSO در این پایگاه داده وجود دارند که حدود ۳۵۰ مورد از آن‌ها در مورد بهبود بخشیدن به عملکرد الگوریتم PSO و تغییر دادن آن نوشته شده‌اند. در حدود ۷۰۰ مقاله، که ۵۵ عدد از آن‌ها مقالات چاپی یا ژورنال هستند، در مورد کاربردهای الگوریتم PSO هستند. در بسیاری از این مقالات، الگوریتم PSO برای استفاده در کاربرد مورد نظر نویسندگان، به نحوی تغییر داده شده است که جواب‌های بهتری به دست بیایند. باقی مقالات در آماری که ارایه خواهد شد، در نظر گرفته نشده‌اند. این تحلیل‌های به صورت یک گزارش فنی به وسیله *ریکاردو پولی*[۱] و در ۲۰ نوامبر سال ۲۰۰۷ ارایه شده است. شناسایی شاخه‌های اصلی کاربردها، به صورت دستی و با کمک گرفتن از کامپیوترهای کوچک انجام گرفته است. فهرست کاربردهای الگوریتم PSO به همراه آمار مربوطه، به صورت طبقه بندی شده در جدول شکل ۵-۵ آمده است [۵۵، ۵۶].

---

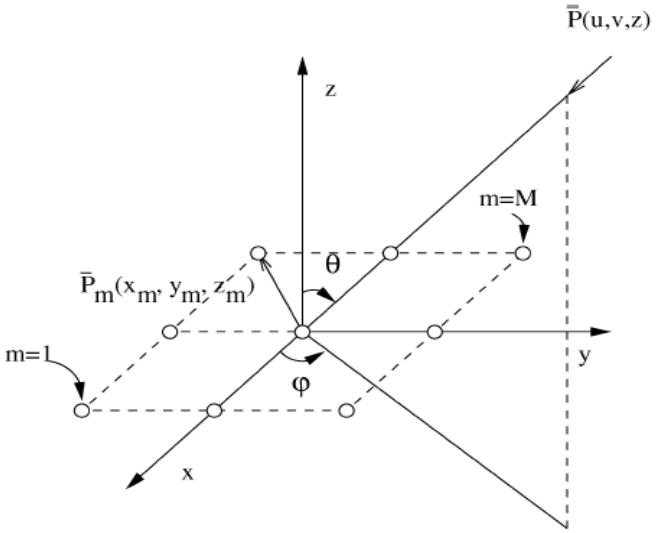
[۱] Riccardo Poli

ردیف	زمینه‌ی کاربردی	تعداد مقالات	درصد مقالات
۱	پردازش تصویر	۵۱	۷/۶
۲	شبکه‌های الکتریکی و توزیع بار	۴۸	۷/۱
۳	کنترل	۴۷	۷/۰
۴	الکترونیک و الکترمغناطیس	۳۹	۵/۸
۵	طراحی آنتن	۳۹	۵/۸
۶	تولید نیرو و سیستم‌های قدرت	۳۹	۵/۸
۷	زمان‌بندی و برنامه‌ریزی	۳۸	۵/۶
۸	طراحی	۳۰	۴/۴
۹	طراحی و بهینه‌سازی شبکه‌های ارتباطی	۳۰	۴/۴
۱۰	زیست‌شناسی و پزشکی	۲۹	۴/۳
۱۱	دسته‌بندی و طبقه‌بندی اطلاعات	۲۹	۴/۳
۱۲	سیستم‌های فازی و فازی-عصبی	۲۶	۳/۸
۱۳	پردازش سیگنال	۲۶	۳/۸
۱۴	شبکه‌های عصبی	۲۶	۳/۸
۱۵	مسایل بهینه‌سازی ترکیبی	۲۴	۳/۵
۱۶	رباتیک	۲۳	۳/۴
۱۷	پیش‌بینی	۲۰	۲/۹
۱۸	مدل‌سازی	۱۹	۲/۸
۱۹	عیب‌یابی و رفع عیوب	۱۶	۲/۳
۲۰	حس‌گرها و شبکه‌های حس‌گری	۱۳	۱/۹
۲۱	گرافیک کامپیوتری	۱۲	۱/۷
۲۲	طراحی و بهینه‌سازی موتورهای مکانیکی و الکتریکی	۱۰	۱/۴
۲۳	متالوژی	۹	۱/۳
۲۴	بازی‌ها و تولید موسیقی	۹	۱/۳
۲۵	امنیتی و نظامی	۹	۱/۳
۲۶	اقتصاد و بازرگانی	۷	۱/۰
	جمع	۷۰۰	۱۰۰

شکل ۵-۵: آمار مربوط به مقالات کاربردی الگوریتم PSO در IEEE Xplore [۵۶].

۵-۶ الگوریتم انبوه ذرات برای کنترل آرایه فازی وفقی

آرایه‌های فازی که در قبل به وسیله الگوریتم ژنتیک برای ایجاد پرتو راه دور مورد نظر بهینه‌سازی شده‌اند، نتایج قابل قبولی را ارایه کرده‌اند. اما در اینجا طراحی به وسیله الگوریتم انبوه ذرات انجام پذیرفته است. در ژنتیک الگوریتم برای دست‌یابی به جوابی بهتر، می‌توان پارامترهای مهمی از قبیل میزان جمعیت[۱]، احتمال برخورد[۲]ها، میزان جهش ژنی[۳] و جهش کروموزنی[۴] را دستکاری کرد. در حالی که در الگوریتم انبوه ذرات، میزان ذرات پراکنده شده[۵] و میزان اینرسی[۶] و شتاب قابل دسترس می‌باشند. بنابراین الگوریتم انبوه ذرات نسبت به الگوریتم ژنتیک به پارامترهای کمتری نیاز دارد که باعث می‌شود تا روند این روش ملموس‌تر از الگوریتم ژنتیک باشد [۵۷].



شکل ۵-۶: نمایش هندسی مساله [۱].

---

[۱] population size
[۲] crossover probability
[۳] gene mutation
[۴] chromosome mutation
[۵] swarm size
[۶] inertial weight

۵-۶-۱ فرمول‌بندی ریاضی

اگر همانند شکل ۵-۶ آرایه‌ای با M عنصر را که با فاصله‌های برابری از هم‌دیگر چیده شده‌اند، داشته باشیم، آنگاه سیگنال مطلوبی که به mامین عنصر این آرایه می‌رسد را می‌توان با رابطه زیر بیان کرد [۵۷]:

$$S_m^d(t) = p_d(t)e^{j\beta_m^d} \quad m = 1, ..., M \qquad (۳۱-۵)$$

در رابطه با لا $\beta_m^d = (2\pi/\lambda)(u_d x_m + v_d y_m + z_d z_m)$ بوده و $u_d = \sin\theta_d \cos\phi_d$، $v_d = \sin\theta_d \sin\phi_d$، $z_d = \cos\theta_d$، و x_m، y_m، z_m مختصات کارتزین mامین عنصر می‌باشند. λ طول موج فضای آزاد و θ_d، ϕ_d متغیرهای مربوط به مختصات کروی بوده و جهت ورود (DOA[۱]) سیگنالی را که دارای پوش $p_d(t)$ می‌باشد را معین می‌کنند. در محیط واقعی، تعداد متغیرهای سیگنال تداخلی که به آرایه وارد می‌شود را با $S_i(t), i = 1, ..., I$ نشان داده و مقدار آن در سنسور عنصر mام با رابطه زیر بیان می‌گردد:

$$S_m^i(t) = p_i(t)e^{j\beta_m^i} \quad \begin{matrix} m = 1, ..., M \\ i = 1, ..., I \end{matrix} \qquad (۳۲-۵)$$

که در آن $\beta_m^i = (2\pi/\lambda)(u_i x_m + v_i y_m + z_i z_m)$ و $p_i(t)$ پوش jامین تداخلی است که دارای فرکانس زاویه‌ای همسان سیگنال مطلوب ω_d می‌باشد. هم‌چنین برای سیگنال‌های تداخلی فرض شده است که دارای باند باریک می‌باشند. و در انتها نویز را با توزیع گوسی و توان $\wp_n$ وارد محاسبات می‌کنیم. با این فرضیات ماتریس کواریانس با ابعاد $M \times M$ را که با سیگنال مطلوب $(\Im \Leftarrow d)$ و یا با iامین سیگنال تداخلی $(\Im \Leftarrow i)$ مرتبط است با رابطه زیر بیان می‌شود:

$$\Phi_\Im = E\left\{\sum_{m=1}^{M}\sum_{n=1}^{M} S_m^{\Im *}(t) S_n^{\Im}(t)\right\} \qquad (۳۳-۵)$$

در رابطه بالا $E(\bullet)$ همان مقدار میانگین یا مقدار مورد انتظار بوده و علامت $*$ به معنی مزدوج مختلط می‌باشد. مقدار ماتریس کواریانس نویز به صورت زیر نمایش داده می‌شود:

$$\Phi_n = p_n 1^M \qquad (۳۴-۵)$$

1^M بیان کننده هویت M بعدی ماتریس می‌باشد.
هم‌چنین ماتریس کواریانس سیگنال نامطلوب در گیرنده برابر با

$$\Phi_u = \sum_{i=1}^{I} \Phi_i + \Phi_n \qquad (۳۵-۵)$$

بوده و توان آن با رابطه زیر بیان می‌گردد:

---

[۱] Direction Of Arriva

$$\wp_u = \frac{1}{2}\underline{W}^{T*}\Phi_u\underline{W} \qquad (۳۶-۵)$$

که در آن T به معنی ترانهاده بوده و W به صورت زیر بیان می‌گردد:

$$\underline{W} = \left\{ w_m e^{j\phi_m};\ m = 1, ..., M \right\} \qquad (۳۷-۵)$$

در رابطه بالا w_m و ϕ_m، ضریب دامنه و شیفت فاز mامین عنصر می‌باشند.
میزان توان دریافتی از سیگنال مطلوب نیز به صورت زیر بیان می‌شود:

$$\wp_d = \frac{1}{2} p_d^2(t)\left|\underline{W}^T \underline{U}(\theta_d, \phi_d)\right|^2 \qquad (۳۸-۵)$$

که در آن $\underline{U}(\theta_d, \phi_d)$ برداری است که mامین عنصر آن برابر با $U_m(\theta_d, \phi_d) = e^{j\beta_m^d}$ می‌باشد.

بنابراین با توجه به روابط (۵-۳۶) و (۵-۳۸) میزان سیگنال مطلوب به مجموع سیگنال تداخل با نویز (SINR[۱]) که با ($SINR$) نشان می‌دهیم با رابطه زیر بیان می‌گردد:

$$\Psi(\underline{W}) \triangleq \frac{\wp_d}{\wp_u} = \frac{p_d^2(t)\left|\underline{W}^T \underline{U}(\theta_d, \phi_d)\right|^2}{\underline{W}^{T*}\Phi_u\underline{W}} \qquad (۳۹-۵)$$

مقدار بیشینه رابطه بالا نسبت به $\underline{W}$، همان هدف مورد نظر می‌باشد. با توجه به این که مقادیر Φ_u و $p_d(t)$ نامعلوند و به طور مستقیم نمی‌توان آن‌ها را اندازه گیری کرد، رابطه (۵-۳۹) موجود نخواهد بود. با این وجود می‌توان مساله را با بیشینه کردن تابع هزینه قابل محاسبه زیر حل نمود.

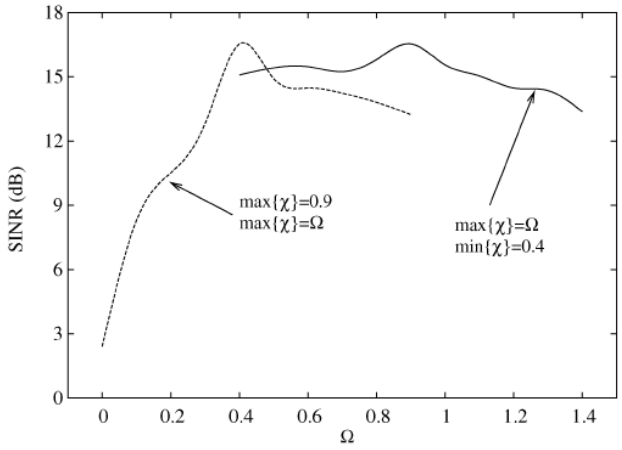
$$f(\underline{W}) = \frac{\left|\underline{W}^T \underline{U}(\theta_d, \phi_d)\right|^2}{\underline{W}^{T*}\Phi_t\underline{W}} \qquad (۴۰-۵)$$

که در آن $\Phi_t = \Phi_d + \sum_{i=1}^{I}\Phi_i + \Phi_n$ کمیتی است که می‌توان آن را در گیرنده اندازه گیری کرد. به علت پیچیدگی و طبیعت تغییر پذیری با زمان رابطه (۵-۴۰) بهتر است تا بهینه‌سازی با روش‌های مناسبی که به دنبال بهینه سراسری هستند، انجام داده شود.

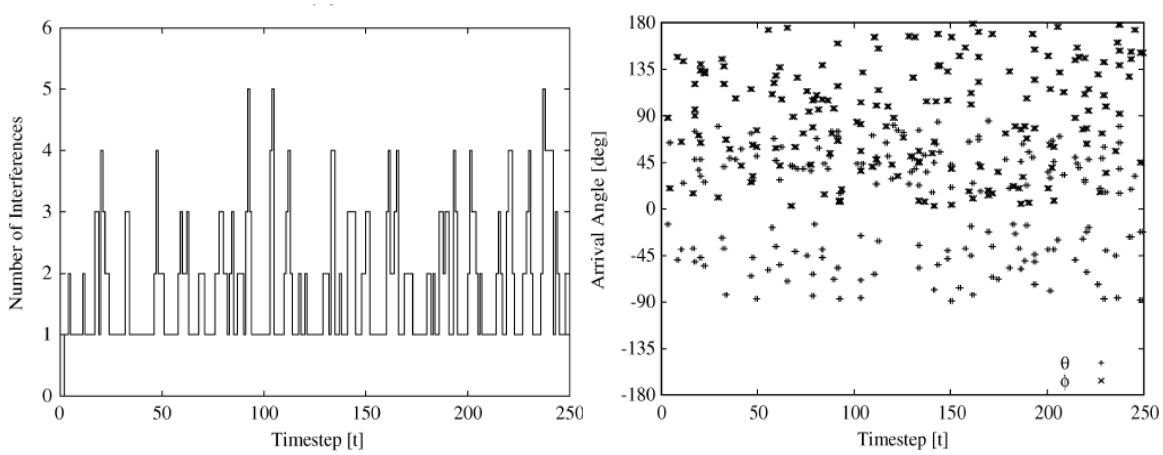
	LMS	***DPA***	***CGA***	***PSO***	***Optimal***
$\alpha v\{SIN\ R\}$	0.87	11.64	14.93	19.61	34.18
$\sigma_{SINR}[\times 10^{-1}]$	22.81	22.25	24.92	21.53	51.08

شکل ۵-۷: کنترل وفقی آرایه خطی ۲۰ عنصری [۱].

[۱] signal-to-interference-plus-noise-ratio



شکل ۵-۸: فرآیند تعیین χ برای آرایه خطی. مقدار میانگین SINR برای مقادیر متفاوت $\chi_{\max}$ با $(\chi_{\min} = 0.4)$ و $\chi_{\min}$ با $(\chi_{\max} = 0.9)$ در گام زمانی $T = 100$ [۱]

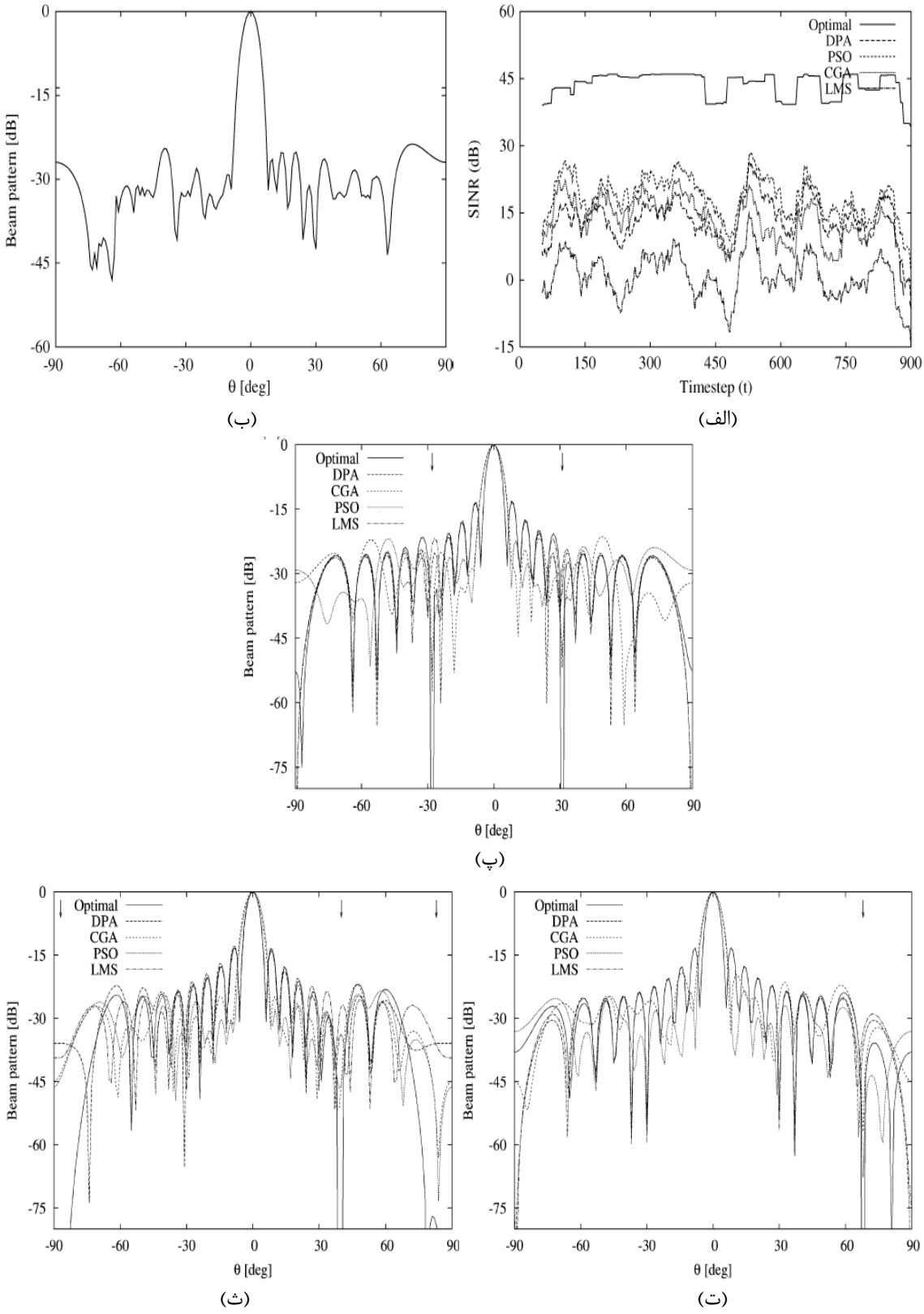


(ب) (الف)

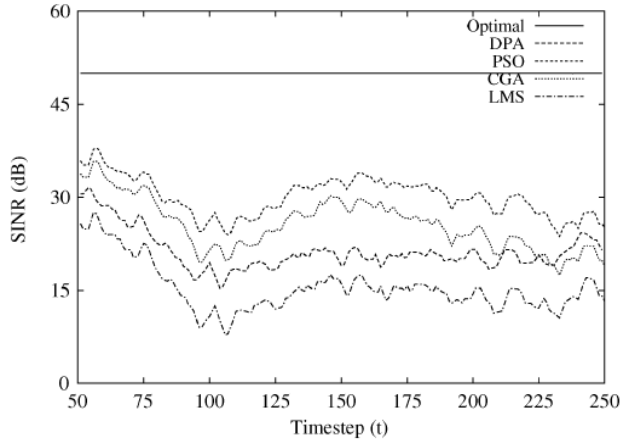
شکل ۵-۹: نمای تداخل. (الف) توزیع زوایای سیگنال تداخلی ورودی بر حسب گام زمانی، (ب) تعداد سیگنال‌های تداخلی در هرگام زمانی [۱]

	LMS	***DPA***	***CGA***	***PSO***	***Optimal***
$\alpha v\{SIN\ R\}$	15.24	21.24	25.59	30.03	49.98
$\sigma_{SINR}[\times 10^{-1}]$	16.32	9.72	16.87	9.74	0.006

شکل ۵-۱۰: کنترل وفقی آرایه صفحه‌ای ۱۰×۱۰ [۱].



شکل ۵-۱۱: کنترل وفقی آرایه خطی ۲۰ المانی (الف) محاسبه میانگین SINR بر حسب k با روش های متفاوت (ب) الگوی پرتو غیر فعال. چند نمونه پرتو سنتز شده در زمان‌های (پ) $t = 396$ (ت) $t = 604$ (ث) $t = 802$ [۱].



شکل ۵-۱۲: کنترل وفقی آرایه صفحه‌ای ۱۰×۱۰. محاسبه مقدار میانگین SINR بر حسب t با استفاده از روش‌های متنوع [۱].

۵-۶-۲ روش بهینه‌سازی

در الگوریتم انبوه ذرات استاندارد که با دامنه اعداد حقیقی سر و کار دارد، مسیر ذرات به گونه‌ای است که تنها برخی از ابعاد تغییر می‌یابند، در حالی که در حالت دودویی مسیرها به صورت احتمالی و با صفر و یک شدن مختصات عمل می‌کنند. صورت دودویی الگوریتم طی چند گام که برای کنترل آرایه فازی بکار می‌رود، توضیح داده خواهد شد. t به عنوان گام زمانی می‌باشد که در هر مرحله از آن تغییری به وجود می‌آید.

- **گام ۰- کدگذاری:** اگر در آرایه فازی mامین عنصر آرایه از طریق یک شیفت دهنده فاز دیجیتالی L بیتی کنترل شود، Pامین جواب آزمایشی دودویی را به صورت زیر می‌نویسیم:

$$\underline{\zeta}^p = \left\{ \varphi_m^{p,l};\ l = 1, ..., L;\ m = 1, ..., M \right\} p = 1, ..., P \qquad (۴۱-۵)$$

که دنباله‌ای از فازهای کوانتیزه شده $\phi_m^p, m = 1, ..., M$ را که با رابطه زیر بیان می‌شوند، کدگذاری می‌کند.

$$\phi_m^p = \left(\frac{\phi_{\max} - \phi_{\min}}{2^L - 1} \right) \sum_{l=1}^{L} 2^{l-1} \varphi_m^{p,l} + \phi_{\min} \qquad (۴۲-۵)$$

در رابطه بالا $\phi_{\max}$ و $\phi_{\min}$ به ترتیب بیان‌گر بیشترین و کمترین مقدار بازه مربوط به پارامتر ϕ_m می‌باشند. منظور از $\varphi_m^{p,l}$، lامین بیت ϕ_m^p کد شده می‌باشد.
با هر بردار موقعیت $\underline{\zeta}^p$، بردار سرعتی به صورت $\underline{v}^p = \left\{ v_m^{p,l};\ l = 1, ..., L;\ m = 1, ..., M \right\}$ تعیین می‌شود، که توانایی ذرات را در تغییر موقعیت‌شان نشان می‌دهد. برای نمونه برداری از فضای جواب‌ها و دست یابی به بهترین جواب، موقعیت ذرات در تکرارهای متوالی از $\underline{\zeta}_k^p$ به $\underline{\zeta}_{k+1}^p$ تغییر

می‌یابد. هم‌چنین، هر $v_m^{p,l}$ احتمال $\varphi_m^{p,l}$ را با مقدار 1 نشان می‌دهد؛ یعنی اگر $v_m^{p,l}$ دارای مقدار بالایی باشد، $\varphi_m^{p,l}$ برابر 1 و در غیر این صورت برابر 0 در نظر گرفته می‌شود.

- **گام ۱- اختصاص مقدار اولیه:** در شروع فرآیند مقدار $(k = 0)$ در نظر گرفته می‌شود. با توجه به عملکرد الگوریتم انبوه ذرات موقعیت اولیه p ذره $\Gamma_0 = \left\{ \underline{\zeta}_0^p ;\ p = 1, ..., P \right\}$ همراه با سرعت اولیه $V_0 = \left\{ \underline{v}_0^p ;\ p = 1, ..., P \right\}$ به طور تصادفی و با استفاده از قوانین زیر تعیین می‌شوند:

$$\varphi_{m,0}^{p,l} = \begin{cases} 1, & if \ \ \rho_{m,0}^{p,l} \geq 0.5 \\ 0, & otherwise \end{cases} \qquad v_{m,0}^{p,l} = \begin{cases} 1, & if \ \ \sigma_{m,0}^{p,l} \geq 0.5 \\ 0, & otherwise \end{cases} \qquad (۴۳-۵)$$

در رابطه فوق $\rho_{m,k}^{p,l}$ و $\sigma_{m,k}^{p,l}$ اعدادی تصادفی میان 0 تا 1 بوده و دارای توزیع یکنواخت می‌باشند.

- **گام ۲- ارزیابی تابع هزینه:** درجه مطلوبیت هر ذره در تکرار kام با محاسبه تابع هزینه $f_k^p = f\left\{ \underline{W}(\underline{\zeta}_k^p) \right\}$ ارزیابی می‌گردد. بهترین موقعیتی که تاکنون به وسیله ذره pام با بهترین موقعیتی که به وسیله گروه حاصل شده است، به ترتیب در متغیرهایی با نام *بهترین ذره قبلی*[۱] $\left(\underline{\xi}_k^p = arg\left(max_{h=1,...,k} \left[f\left\{ \underline{W}(\underline{\zeta}_h^p) \right\} \right] \right) \right.$ و *بهترین سراسری*[۲] $\left. \underline{\varsigma}_k = arg\left(max_{p=1,...,p} \left[f\left\{ \underline{W}(\underline{\xi}_k^p) \right\} \right] \right) \right.$ ذخیره می‌شوند.

- **گام ۳- تجدید تکرار:** تکرار با قرار دادن $k = k + 1$ به پیش برده می‌شود.

- **گام ۴- معیار خاتمه:** اگر به آخرین تکرارها $(k = K)$ برسیم و یا مقدار تابع هزینه از مقدار آستانه $\eta\left(f\left\{ \underline{W}(\underline{\varsigma}_k) \right\} \leq \eta \right)$ کمتر شود، بهینه‌سازی متوقف شده و $\underline{\varsigma}_k$ به عنوان جواب مساله پذیرفته می‌شود. بعلاوه برای بهبود واکنش الگوریتم در مقابل تغییرات محیطی که در میان اجرای گام‌های زمانی رخ می‌دهد و بهره برداری کامل از شباهت‌های موجود میان شرایط مختلف، ذرات مطلوب در یک بافر با طول محدود B ذخیره می‌شوند. عنصرهای بافر B در هر گام زمانی بدین صورت جدید می‌شوند که $\underline{\gamma}_B \perp_t = \underline{\varsigma}_k$ و $\underline{\gamma}_b \perp_t = \underline{\gamma}_{b+1} \perp_{t-1},\ b = 1, ..., B-1$. این B جواب در طی بهینه‌سازی و زمانی که اعتبار سیستم بر حسب (SINR) تنزل می‌یابد، مورد بررسی قرار می‌گیرند. اگر به حالتی که در آن (SINR) کاهش می‌یابد، برخورد نشود، گام پنجم شروع می‌شود.

- **گام ۵- تجدید سرعت:** سرعت هر ذره با توجه به رابطه زیر تغییر می‌یابد:

---

[۱] previous best

[۲] global best

$$v_{m,k}^{p,l} = \begin{cases} v_{\max}, & if\ v_{m,k}^{p,l} > v_{\max} \\ -v_{\max}, & if\ v_{m,k}^{p,l} < -v_{\max} \\ v_{m,k}^{p,l}, & otherwise \end{cases} \qquad (۴۴-۵)$$

که $v_{\max}$ مقدار ثابتی است که به آن مقدار نگه دارنده[۱] گفته می‌شود و همانند میزان جهش در الگوریتم ژنتیک می‌باشد. بر خلاف الگوریتم انبوه ذرات با مقادیر پیوسته که در آن با افزایش مقدار نگه دارنده ناحیه‌ی جستجو نیز وسیع‌تر می‌شود، در حالت دودویی با مقدار $v_{\max}$ کمتر، جهش بالایی رخ می‌دهد. به طور کلی، برای اطمینان از این که همیشه شانسی برای تغییر حالت $\varphi_m^{p,l}$ (از 0 به 1 و برعکس) وجود داشته باشد، $v_{\max}$ برابر با 4.0 در نظر گرفته می‌شود. به علاوه، سرعت ذرات به صورت زیر تعیین می‌شوند.

$$v_{m,k}^{p,l} = \chi v_{m,k-1}^{p,l} + a_1\rho_1\left\{\xi_{m,k}^{p,l} - \varphi_{m,k}^{p,l}\right\} + a_2\rho_2\left\{\delta_m^l - \varphi_{m,k}^{p,l}\right\} \qquad (۴۵-۵)$$

در رابطه بالا، $\underline{\delta} = arg\left(max_{b=1,\ldots,B}\left[f\left\{\underline{W}(\underline{\gamma}_b)\right\}\right]\right)$ بوده و ρ_1 و ρ_2 دو عدد تصادفی مثبتی هستند که دارای توزیع یکنواخت و محدودیت $\rho_1 + \rho_2 = 4.0$ می‌باشند. a_1 و a_2 ثوابتی هستند که به ترتیب آگاهی[۲] و شتاب اجتماعی[۳] خوانده می‌شوند.

❖ **گام ۶- تجدید موقعیت:** جایگاه ذرات با رابطه زیر تغییر می‌کند:

$$\varphi_{m,k}^{p,l} = \begin{cases} 1, & if\ \ \rho_{m,k}^{p,l} < S\left(v_{m,k}^{p,l}\right) \\ 0, & otherwise \end{cases} \qquad (۴۶-۵)$$

در رابطه بالا $S(\bullet)$ تابع سیگموید[۴] است که با رابطه زیر بیان می‌شود:

$$S\left(v_{m,k}^{p,l}\right) = \frac{1}{1 + exp\left(-v_{m,k}^{p,l}\right)} \qquad (۴۷-۵)$$

بنابراین احتمال این که $\varphi_{m,k}^{p,l} = 1$ باشد، برابر با تابع $S\left(v_{m,k}^{p,l}\right)$ و احتمال این که $\varphi_{m,k}^{p,l} = 0$ باشد، برابر با $\left[1 - S\left(v_{m,k}^{p,l}\right)\right]$ می‌باشد. هم‌چنین احتمال تغییر $\varphi_{m,k}^{p,l}$، (از $1 \to 0$ یا $0 \to 1$) برابر با $\left\{S\left(v_{m,k}^{p,l}\right)\left[1 - S\left(v_{m,k}^{p,l}\right)\right]\right\}$ می‌باشد.

سپس به گام دوم برگشته می‌شود.

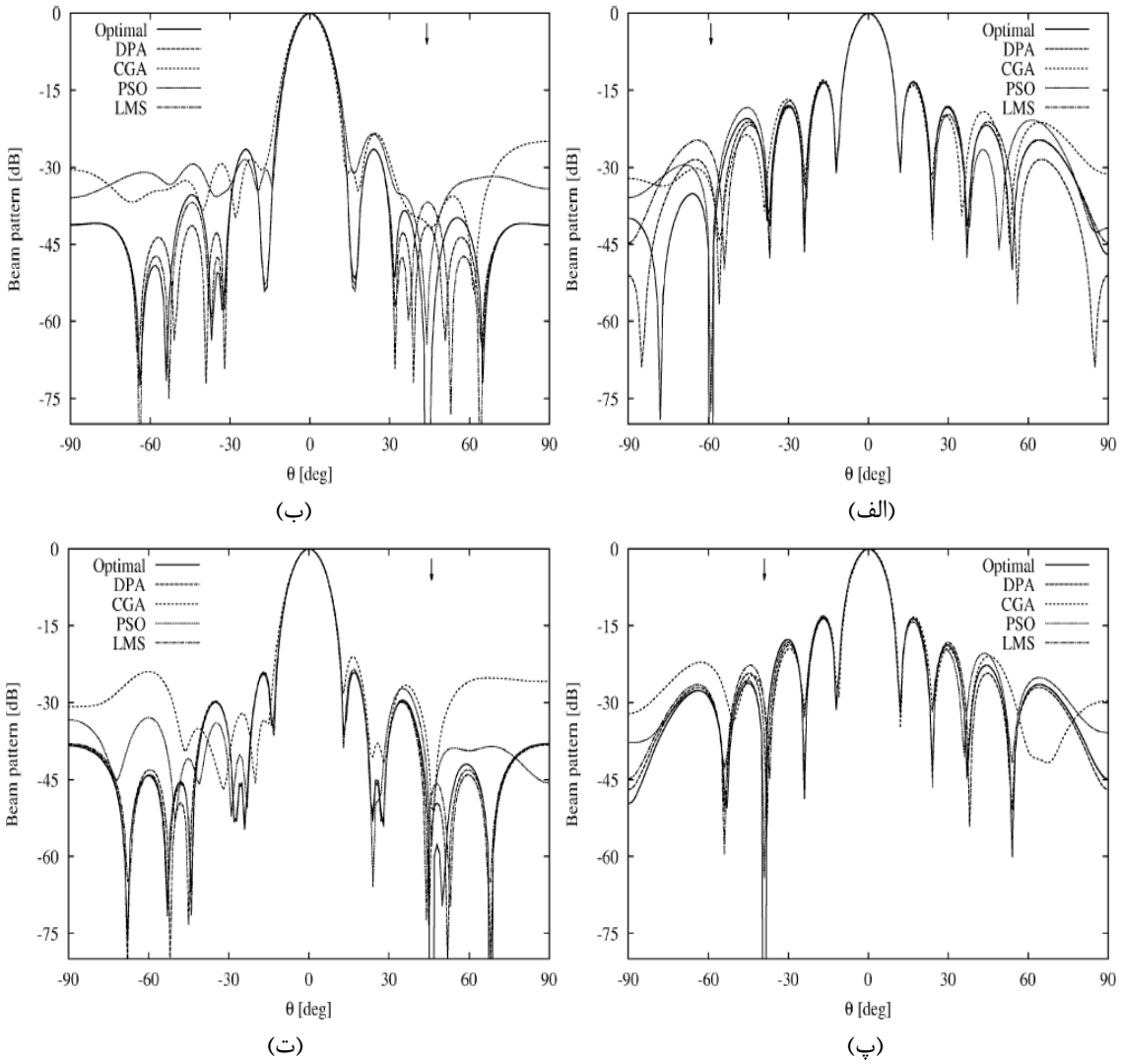
---

[۱] clamping value

[۲] cognition

[۳] social acceleration

[۴] sigmoid



شکل ۵-۱۳: کنترل وفقی آرایه صفحه‌ای ۱۰×۱۰ . چند نمونه پرتو سنتز شده در زمان‌های (الف) $t = 55$ $(\phi = 173^\circ)$، (ب) $t = 105$ $(\phi = 132^\circ)$، (پ) $t = 205$ $\phi = 8^\circ$، (ت) $\phi = 60^\circ$ [۱].

۵-۶-۳ نتایج عددی

در این قسمت، با ارایه نتایج انتخاب شده‌ای از تجربیات عددی، توانایی الگوریتم انبوه ذرات برای کنترل آرایه‌ها تخمین زده می‌شود [۱]. برای تایید نتایج عددی، روش پیشنهادی بر روی آرایه‌های خطی و صفحه‌ای بررسی می‌گردد. در بررسی پارامترهایی همچون نویز گوسی پیش زمینه که $30dB$ کمتر از سیگنال مطلوب (که با زاویه $\theta_d = 0^\circ$ وارد می‌شود) و تداخل دامنه $30dB$ بالای سیگنال مطلوب تعیین می‌گردد و همانند آن‌چه در عمل وجود دارد، می‌باشد. در آزمایشات، سیگنال مطلوب و نویز ناهمبسته در نظر گرفته شده‌اند.

نتایج حاصل از الگوریتم انبوه ذرات با روش‌های فنی دیگری همچون (الف) روش بهینه اپلبام[۱] (که در آن دامنه‌ها و فازهای آرایه‌ها به طور هم‌زمان تنظیم می‌شوند)، (ب) نسخه اصلاح شده روش اپلبام که در آن فازهای اپلبام کوانتزه می‌شوند (DPA)، (پ) نسخه سفارشی الگوریتم ژنتیک (CGA) و (ت) الگوریتم میانگین حداقل مربعات (LMS)، مقایسه شده است. برای کنترل آرایه‌ها وزن آرایه‌ها به صورت $w_m = 1,\ m = 1, ..., M$ و فاز آن‌ها با استفاده از شیفت دهنده فاز دیجیتالی $N_{bit} = 6$ بیتی در بازه $\phi_{\min} = 0^\circ$ تا $\phi_{\max} = 360^\circ$ به طور مکرر تنظیم می‌شود.

با استفاده از روش‌های کنترلی چند عامله نظیر الگوریتم ژنتیک و الگوریتم انبوه ذرات نتایج آزمایشی با $P = M$ جمعیت در تکرار $K = 20$ به دست می‌آیند. با این روش مقادیری به طور ابتکاری برای پارامترهایی نظیر $a_1 = a_2 = 2.0$، $B = P/10$ و χ که در حین اجرا به طور خطی از 0.9 تا 0.4 تغییر می‌کند، به دست می‌آید. به عنوان مثالی شکل ۵-۸ نتایج کالیبراسیون χ را با توجه به هندسه مربوطه مشخص می‌سازد.

۵-۶-۳-۱ آرایه خطی

در این قسمت به بررسی آرایه خطی $M = 20$ عنصری که از هم‌دیگر نیم طول موج فاصله دارند، پرداخته می‌شود. نتایجی که بر حسب (SINR) بیان می‌شوند با تحقق بخشیدن فرآیندهای آماری نویز و تداخل بعد از 50 بار اجرای برنامه به دست آمده‌اند. بنابراین بعد از اجراهای مکرر، فرض شده که جهت سیگنال اغتشاش[۲] دارای توزیع یکنواخت بوده و زمان ورود آن با توزیع پواسن با $\lambda = 1$ و دو گام زمانی برای زمان بقا، مدل می‌شود.

شکل ۵-۹ (الف) نمونه‌ای از زاویه ورودی سیگنال‌های تداخلی را در هر گام زمانی نشان می‌دهد. برای تکمیل موضوع، تعداد سیگنال‌های تداخلی دریافت شده در هر گام زمانی در شکل ۵-۹ (ب) نمایش داده شده است. رفتار (SINR) به عنوان تابعی از گام زمانی در شکل ۵-۱۱ (الف) آمده است. در این شکل نتایج حاصله از الگوریتم انبوه ذرات با روش‌های دیگر مقایسه شده است.

همان‌طور که از اشکال دیده می‌شود، الگوریتم انبوه ذرات نتایج بهتری را نشان می‌دهد. میانگین (SINR) در تمامی نسل‌ها و شبیه‌سازی‌ها برای الگوریتم انبوه ذرات برابر $av\{\text{SINR}\}|_{\text{PSO}} = 19.61dB$ و برای ژنتیک الگوریتم سفارشی شده $av\{\text{SINR}\}|_{\text{CGA}} = 14.93dB$ می‌باشد (جدول شکل ۵-۷). این اختلاف در حدود $4dB$، به خصوص هنگامی که بسیاری از نقاط دو گراف در شکل ۵-۱۱ (الف) در بازه [7 dB, 22 dB] قرار می‌گیرند، بسیار مهم می‌باشد.

---

[۱] Applebaum

[۲] jamming

شکل‌های ۵-۱۱ (پ) و ۵-۱۱ (ث) برخی از موارد را برای پرتو راه دور که از پرتو غیر فعال[۱] شکل ۵-۱۱ (ب) شروع می‌شوند، روشن می‌سازد. زمان‌هایی که در آن‌ها اغتشاش وارد می‌شود را می‌توان در نتایج حاصله دید. برای جزئیات بیشتر این اشکال موارد زیر را بیان می‌کنند: دو سیگنال اغتشاش در $\theta = -28^\circ$ و $\theta = 30^\circ$ در شکل ۵-۱۱ (پ) در $t = 396$ و یک سیگنال تداخلی در زاویه $\theta = 68^\circ$ در شکل ۵-۱۱ (ت) در $t = 604$ و سه اغتشاش انداز در زوایای $\theta = 40^\circ$، $\theta = -86^\circ$ و $\theta = 83^\circ$ در زمان $t = 802$ در شکل ۵-۱۱ (پ) دیده می‌شود. به طور کلی روش‌های چند عامله همچون الگوریتم انبوه ذرات و الگوریتم ژنتیک سفارشی شده به سرعت در پرتو راه دور صفرهایی ایجاد می‌کنند که با ایجاد آشفتگی بسیار کم در لوب اصلی باعث تخطی عملکرد فن‌های غیر احتمالی و مطلوب دیگر می‌شود. صفرهای عمیقی که در پرتو حاصله از الگوریتم انبوه ذرات به‌دست آمده، باعث بهبود نتایج (SINR) شده است.

۵-۶-۳-۲ آرایه‌های صفحه‌ای

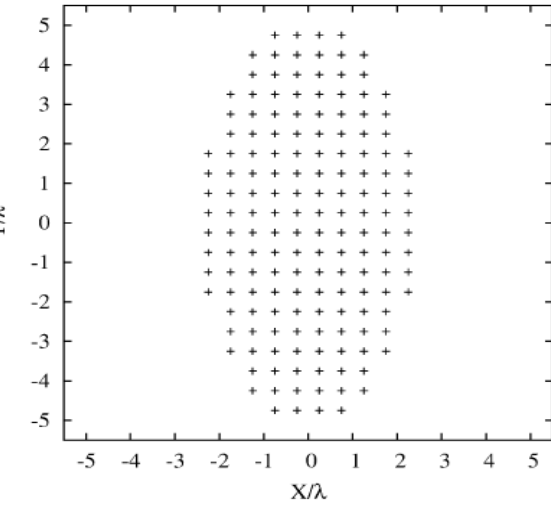
این آزمایش به یک آرایه صفحه‌ای $M = 10 \times 10$ مربوط می‌شود. در این آرایه فاصله عنصرها از هم‌دیگر نصف طول موج می‌باشد.

با توجه به مختصات زاویه‌ای که اغتشاش اندازها دارند، با همان مدل آماری مثال قبلی و شکل ۵-۹ (الف) مدل می‌شوند. شکل ۵-۱۲ رفتار (SINR) میانگین را طی $T = 250$ گام زمانی نشان می‌دهد. همان‌طوری که در جدول شکل ۵-۱۰ نشان داده شده الگوریتم انبوه ذرات عملکرد بالایی را نسبت به فن‌های دیگر از خود نشان می‌دهد. برای جزئیات بیشتر با بالا بردن میانگین (SINR) به اندازه $5dB$ برای ژنتیک الگوریتم سفارشی شده (CGA) و در حدود $10dB$ برای روش اپلبام اصلاح شده (DPA)، می‌توان به عملکرد الگوریتم انبوه ذرات دست یافت. به عنوان مثالی شکل ۵-۱۳ الگوی پرتو را در گام‌های زمانی متفاوت نشان می‌دهد. (الف) در $t = 55$ اغتشاش از زاویه $\theta_1 = -59^\circ$، $\phi_1 = 173^\circ$ ، (ب) در $t = 105$ اغتشاش از زاویه $\theta_1 = 43^\circ$، $\phi_1 = 132^\circ$ و در $t = 205$ (پ) از زاویه $\theta_1 = -39^\circ$، $\phi_1 = 8^\circ$ و (ت) از زاویه $\theta_1 = 46^\circ$، $\phi_1 = 60^\circ$ وارد آرایه می‌شود. با مقایسه روش‌های مختلف و از آن‌چه که از شکل ۵-۱۲ ظاهر می‌شود، الگوریتم انبوه ذرات قادر است تا صفرها را دقیقا در روی سیگنال تداخلی متناظر بیندازد در حالی که روش‌های دیگر دچار خطای چند درجه‌ای می‌شوند. به عنوان مثال این خطا برای $\Delta\theta|_{\text{CGA}} \simeq 5^\circ$ و $\Delta\theta|_{\text{LMS}} \simeq 9^\circ$ می‌باشد.

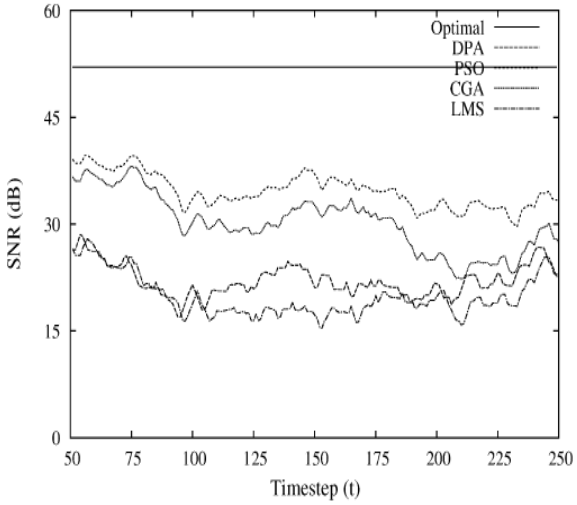
برای بررسی بیشتر روش پیشنهادی، یک آرایه بیضوی $M = 160$ عنصری ارایه شده در شکل ۵-۱۴ تحلیل شده است. دوباره، الگوریتم انبوه ذرات توانایی بالایی را در مقابل روش‌های دیگر از خود نشان داده که نتایج برای بررسی در شکل ۵-۱۵ آمده است. مقادیر میانگین (SINR)، برای روش‌های متفاوت در جدول شکل ۵-۱۶ آمده است.

---

[۱] quiescent



شکل ۵-۱۴ : چیدمان آرایه بیضوی [۱].



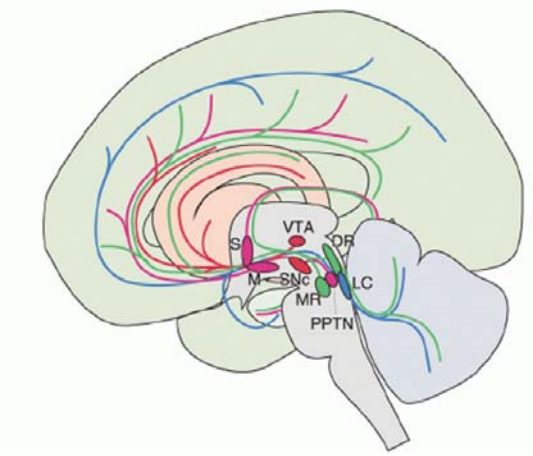
شکل ۵-۱۵ : محاسبه مقدار میانگین SINR بر حسب t با استفاده از روش‌های متنوع [۱].

	LMS	***DPA***	***CGA***	***PSO***	***Optimal***
$\alpha v\{SIN\ R\}$	19.78	20.07	30.17	35.69	52.02
$\sigma_{SINR}[\times 10^{-1}]$	8.30	5.02	16.43	5.59	0.005

شکل ۵-۱۶: کنترل وفقی آرایه صفحه‌ای ۱۰×۱۰ [۱].

۵-۷ خلاصه فصل

در این فصل در ابتدا الگوریتم بهینه‌سازی انبوه ذرات به همراه پارامترهای آن شرح داده شد. پس از آن دو نسخه از نسخه‌های تغییر یافته این الگوریتم از قبیل الگوریتم بهینه‌سازی انبوه ذرات دودویی و الگوریتم بهینه‌سازی انبوه ذرات فازی معرفی شدند. سپس کاربردهای این الگوریتم در مسایل مختلف شرح داده شدند. در نهایت کاربرد این الگوریتم برای کنترل آرایه فازی وفقی توضیح داده شد. در ادامه در فصل آتی مبحث یادگیری تقویتی که معرفی خواهد شد.



فصل ششم

یادگیری تقویتی*

۶-۱ مقدمه

یادگیری تقویتی (RL[۱]) در یک بیان کلی یعنی با توجه به شناختی که از محیط وجود دارد و بر اساس نتایجِ تعاملاتی که با محیط به وجود می‌آید و سودها و زیان‌هایی که در نتیجه انجامِ عمل‌های مختلف به دست آمده، یک استراتژی پیدا شود که با عمل به آن در بلند مدت مطلوبیت خود را بیشینه کند. هدف اصلی در این فصل آشنایی با ادبیات یادگیری تقویتی و برنامه‌نویسی پویا به عنوان یکی از راه‌حل‌های آن می‌باشد. در ابتدا پارامترهای شکل دهنده مساله و سپس ساختار ریاضی آن مانند تابع ارزش، معادله بلمن و غیره بیان شده است. در نهایت روش به دست آوردن سیاست (استراتژی) بهینه توضیح داده شده است [۵۸].

۶-۲ عامل و محیط

یادگیری تقویتی را می‌توان به صورت ساده در یک جمله بیان کرد: «یادگیری به وسیله تعامل با محیط به منظور رسیدن به هدفی مشخص» [۵۸]. تصمیم گیرنده و کسی که یاد می‌گیرد **عامل**[۲] نامیده می‌شود. چیزی که عامل با آن تعامل[۳] می‌کند (که در واقع هر چیزی خارج از عامل را در بر می‌گیرد)، محیط[۴] نامیده می‌شود. این تعامل به صورت پیوسته رخ می‌دهد. بدین ترتیب که عامل تصمیم می‌گیرد

---

* مطالب این فصل ترجمه و برگرفته شده از فصول سوم و چهارم مرجع [۵۸] می‌باشد.

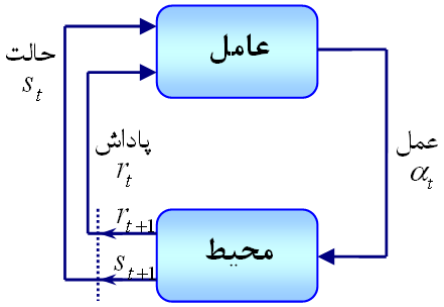
۱ Reinfocement Learning

۲ agent

۳ interact

۴ environment

و بر مبنای آن عملی[۱] که انجام می‌دهد و محیط نیز درپاسخ به این عمل به او پاداشی داده و به حالت[۲] جدیدی می‌رود. به بیان دقیق‌تر، عامل و محیط به صورت متوالی در طی گام‌های زمانی t = 0, 1, 2, 3, … با یک‌دیگر تعامل می‌کنند. در هر گام به طور مقال در گام t، عامل حالت جدیدی از محیط دریافت می‌کند، $s_t \in S$ که S مجموعه حالت‌های ممکن برای محیط است و بر مبنای این حالت، عمل خود $a_t \in A(s_t)$ را انجام می‌دهد که $A(s_t)$ مجموعه عمل‌های ممکنی است که عامل می‌تواند در حالت s_t انجام دهد. یک گام بعد یعنی در t+1، محیط یک پاداش عددی[۳] $r_{t+1} \in R$ بر حسب عمل او در گام قبل، به وی می‌دهد و عامل نیز خود را در حالت جدیدی s_{t+1} می‌یابد. شکل ۶-۱ تعامل محیط و عامل را نشان می‌دهد.



شکل ۶-۱: تعامل محیط و عامل [۵۸].

سیاست یا استراتژی عامل[۴]، π، تابع احتمالی است که احتمال انتخاب شدن هر عمل را در هر حالت و با توجه به گام زمانی می‌دهد. به طور مثال $\pi_t(s, a) = p$ می‌گوید که اگر عامل در زمان t در حالت s قرار گرفته باشد، با احتمال p عمل a را انتخاب می‌کند. روش‌های یادگیری تقویتی نشان می‌دهند که چگونه یک عامل بر اساس تجربه‌ای که از تعامل با محیط به دست می‌آورد، سیاست خود را تغییر می‌دهد. اگر هدف عامل به صورت نادقیق بیان شود عبارت است از بیشینه کردن پاداش‌هایی است که در بلند مدت عامل به دست آورده است.

مثال (الف) ربات[۵] قوطی جمع کن: روباتی را در نظر بگیرید که با یک باطری قابل شارژ کار می‌کند و وظیفه دارد قوطی‌های خالی کنسرو را از روی زمین جمع کرده و درون سبدی که همراه خود دارد قرار دهد. در این‌جا حالت‌های ممکن برای روبات عبارت است از: (۱) شارژ باطری کامل است. (۲) شارژ باطری ضعیف است. (۳) باطری شارژ ندارد و نیز عمل‌های ممکن عبارتند از: (۱) دنبال قوطی‌های خالی بگردد. (۲) در یک‌جا بایستد و منتظر شود کسی قوطی خالی را درون سبد بیندازد. (۳) به خانه برگشته و باطری خود را شارژ کند.

---

۱ action
۲ state
۳ numerical reward
۴ agents policy/stratrgy
۵ robot

۶-۳ اهداف و پاداش[۱]

در یادگیری تقویتی، هدف عامل در قالب سیگنال پاداشی که از محیط دریافت می‌کند، بیان می‌شود. در هر مرحله زمانی، این پاداش به‌صورت عددی ساده بیان می‌شود، $r_t \in R$. در بیانی ساده، هدف عامل، بیشینه کردن مجموع این پاداش‌هاست. دقت شود بیشینه کردن پاداش در بلند مدت مدنظر است و این فقط به معنی بیشینه کردن پاداش در هر مرحله نیست.

مثال (ب): در مثال (الف) برای این که روبات یاد بگیرد قوطی‌های خالی را جمع آوری کند، می‌توان پاداش را برای او بدین‌گونه تعریف کرد که در صورت جمع آوری هر عدد قوطی امتیاز ۱+ می‌گیرد و در غیر این صورت امتیاز او صفر خواهد بود.

مثال (پ): برای روباتی که می‌خواهد شطرنج بازی کردن را یاد بگیرد، به او برای پیروزی یا باخت یا تساوی به ترتیب امتیازهای ۱+ و ۱- و 0 اختصاص داده می‌شود.

نکته مهمی که باید به آن توجه شود این است که باید پاداش به گونه‌ای اختصاص داده شود که عامل با بیشینه کردن آن هدف ما را تامین سازد و نباید به او یاد داد که چگونه هدف را برآورده سازد. به طور مثال در مثال (پ)، روبات تنها باید در صورت پیروزی پاداش بگیرد و اگر برای زیر اهدافی[۲] مانند از بازی خارج کردن مهره‌های حریف به او امتیاز دهیم، ممکن است روبات راهی را یاد بگیرد که به وسیله آن بیشتر مهره‌های حریف را از بازی خارج کند، حتی اگر به قیمت واگذار کردن بازی باشد. در واقع سیگنال پاداش کانال ارتباطی شما با عامل است که به وسیله آن به او می‌گویید به چه هدفی برسد نه این که چگونه به هدف برسد.

حال فرمول کردن مفهوم پاداش شرح داده می‌شود. اگر ترتیب پاداش‌هایی که عامل بعد از مرحله tام می‌گیرد به صورت $r_{t+1}, r_{t+2}, r_{t+3}, \ldots$ باشد، وی به‌دنبال بیشینه کردن امید ریاضی پاداش کل[۳] خواهد بود که پاداش کل به‌صورت جمع پاداش‌های هر مرحله تعریف می‌شود:

$$R_t = \sum_{k=1}^{T-t} r_t + k \qquad (۱-۶)$$

که T آخرین مرحله می باشد.

فرمول فوق برای مسایلی مفید است که فرآیند در مرحله مشخصی پایان پذیرد و به اصطلاح دارای حالت پایانی[۴] باشد. اما مسایل فراوانی وجود دارد که تعامل عامل با محیط تا بی‌نهایت وجود دارد که در

---

۱ goals and rewards
۲ subgoals
۳ return
۴ terminating state

این صورت فرمول فوق با قراردادن $T = \infty$ واگرا می‌شود. برای حل این مشکل، از نرخ تنزیل[۱] به‌صورت زیر استفاده می‌شود:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \qquad (۲-۶)$$

که نرخ تنزیل، $0 \le \gamma \le 1$ می‌باشد

در واقع نرخ تنزیل، ارزش فعلی پاداش‌های آینده را مشخص می‌کند. ارزش واقعی پاداشی که k مرحله دیرتر دریافت شود γ^{k-1} برابرارزش اسمی آن است.

۶-۴ خاصیت مارکوف[۲]

در یادگیری تقویتی، عامل بر اساس سیگنالی که از محیط دریافت می‌کند که به آن حالت محیط می‌گویند، تصمیم می‌گیرد. منظور از واژه «حالت[۳]» هرگونه اطلاعاتی است که در دسترس عامل است. فرض کنید سیگنال s_t نشان دهنده حالت محیط در لحظه t است. این سیگنال علاوه بر این که حاوی اطلاعات موجود در محیط در لحظه t است، می‌تواند اطلاعاتی بیشتر از آن نیز به ما بدهد. حالت محیط می‌تواند از پروسس کردن اطلاعات دریافتی از محیط در همان لحظه یا حتی لحظه‌های قبل به وجود آید. به طور مثال یک روبات با شنیدن واژه «بله» در مرحله t می‌تواند خود را در حالات متفاوتی بسته به سوالی که در مرحله قبل یعنی t-1 از او شده است، بیابد و یا با داشتن موقعیت مکانی خود در لحظه t و t - 1 که می‌تواند سرعت را در لحظه t (که یکی از عامل‌های دخیل در حالت روبات است) به‌دست آورد. پس در همه این موارد، حالت محیط از اطلاعات دریافتی در همان لحظه یا لحظات قبل حاصل می‌شود.

مطلوب ما این است که s_t تمام اطلاعات مفید مربوط به حال و گذشته را در خود خلاصه کند. برای نیل به این هدف چیزی فراتر از یک درک آنی[۴] لازم است اما به مفهوم داشتن تمام تاریخ گذشته نیز نمی‌باشد. به سیگنال حالتی که چنین باشد، گفته می‌شود دارای خاصیت مارکوف است. به‌عنوان مثال چیدمان مهره‌ها روی صفحه شطرنج دارای خاصیت مارکوف است. زیرا با وجود این که تمام حرکت‌هایی که از اول بازی تا حال شده است را به بازیگر نمی‌دهد اما تمام اطلاعات مفید برای ادامه بازی را در خود دارد. یا این که مکان و سرعت یک پرتابه تمام آن چیزی است که لازم است داشته باشد تا به وسیله آن بتوان ادامه حرکت را پیش بینی کرد. پس سیگنالی که حاوی مکان و سر عت باشد با وجود این که مکان و سرعت پرتابه را در لحظه‌های قبل به ما نمی‌دهد اما دارای خاصیت مارکوف است.

---

۱ discount factor
۲ the markov property
۳ state
۴ immediate sensation

اکنون فرمول کردن این خاصیت برای مساله یادگیری تقویتی شرح داده می‌شود. برای این که مساله از لحاظ ریاضی ساده باشد (در واقع بتوان به جای انتگرال و تابع چگالی احتمال از سیگما استفاده کرد)، فرض می‌شود تعداد متناهی از حالت‌ها و پاداش‌ها وجود دارد. فرض شود در گام زمانی t بوده و عمل a_t انجام داده می‌شود. در اکثر سیستم‌های سببی پاسخی که محیط به ما می‌دهد بستگی به تمام آن‌چه که در قبل اتفاق افتاده یا به‌عبارت دیگر به کل تاریخچه[۱] سیستم بستگی دارد. پس پویایی سامانه در صورت معلوم بودنِ کل توزیع احتمال زیر برای تمام مقادیر $s', r, s_t, a_t, r_t, ... r_1, s_0, a_0$ مشخص خواهد شد:

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\} \qquad (۳-۶)$$

اگر سیگنال حالت دارای خاصیت مارکوف باشد، پاسخ محیط در زمان t+1 تنها به حالت محیط و عمل انجام گرفته در زمان t بستگی خواهد داشت. یعنی پویایی سامانه با معلوم بودن توزیع احتمال زیر برای تمام مقادیر ممکن مشخص خواهد شد:

$$\Pr\{s_{t+1} = s', r_{t+1} = r \mid s_t, a_t\} \qquad (۴-۶)$$

اگر رابطه (۶-۳) برای تمامی مقادیر $s', r, s_t, a_t, r_t, ..., r_1, s_0, a_0$ برابر با رابطه (۶-۴) باشد گوییم محیط دارای خاصیت مارکوف است. حال با استفاده از رابطه (۶-۴) و با داشتن حالت و عمل در زمان t می‌توان حالت محیط و پاداش انتظاری[۲] در زمان $t + 1$ را پیش بینی کرد. علاوه بر این با استفاده بازگشتی از رابطه (۶-۴) و با داشتن حالت و عمل در زمان t می‌توان تمامی حالات و پاداش‌های انتظاری در زمان‌های بعد را نیز به دست آورد.

۶-۵ فرآیند تصمیم‌گیری مارکوف[۳]

به مساله یادگیری تقویتی که در آن خاصیت مارکوف برقرار باشد، فرآیند تصمیم‌گیری مارکوف (MDP) گویند. اگر فضای حالات و مجموعه عمل‌های ممکن متناهی باشد، به فرآیند تصمیم گیری مارکوف متناهی (Finite MDP) گویند. یک MDP به وسیله مجموعه حالات و عمل‌های ممکن و احتمال گذر مشخص می‌شود. منظور از احتمال گذر احتمال رفتن به حالت $s_{t+1} = s'$ است اگر در زمان t و در حالت $s_t = s$، عمل $a_t = a$ انجام داده شود:

$$P_{ss'}^{a} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\} \qquad (۵-۶)$$

---

۱ history
۲ expected reward
۳ markov decision process

به‌طور مشابه اگر حالت فعلی و عمل فعلی و نیز حالت بعدی محیط وجود داشته باشد، امید ریاضی پاداش انتظاری درمرحله بعد به صورت زیر خواهد بود:

$$R_{SS'}^{a} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\} \qquad (۶-۶)$$

این دو مقدار $P_{ss'}^{a}$ و $R_{ss'}^{a}$، تمام جنبه‌های مهم پویایی محیط را درMDP مشخص می‌کنند.

مثال (ت) مثال (الف) را در نظر بگیرید. این مثال به یک مساله MDP تبدیل می‌شود. فرض کنید پویایی محیط به این صورت است که بهترین راه برای جمع آوری قوطی‌ها، جستجو برای یافتن آن‌ها است ولی این کار شارژ باطری را ضعیف می‌کند در حالی‌که اگر صبر کند تا کسی قوطی خالی را درون آن بیندازد، شارژ باطری به قوت خودباقی می‌ماند. نیز در هنگام جستجو این امکان وجود دارد که شارژ باطری تمام شده و روبات خاموش شود. روبات تصمیم‌های خود را تنها بر اساس سطح شارژ باطری می‌گیرد. پس فضای حالات ممکن را می‌توان S ={high, low} و فضای عمل‌های ممکن را می‌توان A ={search, wait, recharge} در نظر گرفت.

هنگامی‌که سطح شارژ بالاست، شارژ کردن دوباره باطری بی معناست. پس مجموعه عمل‌های ممکن در هر حالت به صورت زیر است:

A (high) = {search, wait}

A (low) = {search, wait, recharge}

اگر سطح شارژ باطری بالا باشد و یک دوره جستجو انجام شود، با احتمال α سطح شارژ باطری بالا می‌ماند در غیر این صورت سطح شارژ پایین می‌آید. از طرف دیگر اگر سطح شارژ پایین باشد و یک دوره جستجو انجام شود، با احتمال β سطح شارژ باطری پایین می‌ماند در غیر این صورت شارژ باطری تمام می‌شود. در این حالت کسی باید باطری را دوباره شارژ کند تا سطح شارژ آن بالا شود. هر قوطی خالی که روبات جمع کند ۱+ امتیاز و اگر نیاز به کسی پیدا کرد تا آن را شارژ کند ۳- امتیاز می‌گیرد. نیز فرض کنید ($R^{search} \succ R^{wait}$) R^{search}, R^{wait} به ترتیب امید ریاضی تعداد قوطی‌هایی است که روبات در صورت جستجو کردن یا منتظر شدن، جمع آوری می‌کند. این سیستم یک MDP متناهی است و احتمال گذر و پاداش انتظاری آن در شکل ۶-۲ نمایش داده شده است.

$s = s_t$	$s' = s_{t+1}$	$a = a_t$	$P^a_{ss'}$	$R^a_{ss'}$
High	High	Search	α	R^{Search}
High	Low	Search	$1-\alpha$	R^{Search}
Low	High	Search	$1-\beta$	-3
Low	Low	Search	β	R^{Search}
High	High	Wait	1	R^{Wait}
High	Low	Wait	0	R^{Wait}
Low	High	Wait	0	R^{Wait}
Low	Low	Wait	1	R^{Wait}
Low	High	Recharge	1	0
Low	Low	Recharge	0	0

شکل ۶-۲ : احتمال گذر و پاداش انتظاری یک سیستم MDP متناهی [۵۸].

۶-۶ تابع ارزش[۱]

تقریبا تمامی الگوریتم‌های یادگیری بر پایه تخمین تابع ارزش حالت بنا می‌شود. به طور نادقیق می‌توان ارزش حالت را معیاری از خوب بودن آن حالت در نظر گرفت. ارزش حالت را می‌توان این‌گونه توضیح داد که امید ریاضی پاداش کلی است که عامل با شروع از حالت $s_t = s$ و در پیش گرفتن سیاست π به دست می‌آورد. بدیهی است که تابع ارزش با توجه به یک سیاست خاص تعیین می‌شود:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s\right\} \qquad (۶-۷)$$

که $E_{\pi}\{\}$ نشان دهنده امید ریاضی است در صورتی‌که از سیاست π پیروی شود. واضح است که ارزش حالت پایانی[۲] صفر خواهد بود. ما V^{π} را تابع ارزش حالت برای سیاست π نامیده می‌شود. به طور مشابه تابع دیگری به نام $Q_{\pi}(s, a)$ تعریف می‌شود که بیان‌گر امید ریاضی پاداش کلی است که عامل با شروع از حالت $s_t = s$ انجام عمل $a_t = a$ و سپس در پیش گرفتن سیاست π به دست می‌آورد:

$$Q^{\pi}(s, a) = E_{\pi}\{R_t \mid s_t = s, a_t = a\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\right\} \qquad (۶-۸)$$

تابع Q^{π}، تابع ارزش عمل برای سیاست π نامیده می‌شود.

---

۱ value function
۲ terminal state

تابع ارزش حالت و تابع ارزش عمل می‌توانند به صورت تجربی به دست آیند. به‌عنوان مثال اگر عامل سیاست π را در پیش بگیرد و از حالت $s_t = s$ شروع کند، پاداش کلی که با شروع از این حالت به دست می‌آید به عنوان یک مقدار نمونه از متغیر تصادفی $V^{\pi}(s)$ خواهد بود که به وسیله $v^{\pi}(s)$ نشان داده می‌شود. حال اگر این عمل بی‌نهایت بار تکرار شود؛ یعنی، بی‌نهایت نمونه $v^{\pi}(s)$ به‌دست آید و سپس از این نمونه‌ها میانگین بگیریم، این میانگین به مقدار $V^{\pi}(s)$ میل می‌کند. همین روش برای محاسبه $Q^{\pi}(s,a)$ به کار می‌رود به چنین روش‌های تخمین از تابع ارزش، روش مونت کارلو[۱] گویند، زیرا از بی‌شمار نمونه تصادفی از پاداش‌های کلی میانگین‌گیری می‌کند. روشن است که اگر تعداد حالات زیاد باشد، این روش عملی نخواهد بود.

یک خاصیت بنیادینِ تابع‌های ارزش که در یادگیری تقویتی و برنامه‌نویسی پویا[۲] استفاده می‌شود، این است که در یک رابطه بازگشتی صدق می‌کنند. برای هر سیاست π و هر حالت s رابطه زیر بین ارزش s و ارزش حالات وقوع یافته‌ی بلافاصله بعد از آن[۳] برقرار است:

$$V^{\pi}(s) = E_{\pi}\{R_t \mid s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1} \mid s_t = s\right\} \qquad (۹\text{-}۶)$$

$$= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2} \mid s_t = s\right\}$$

$$= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma E_{\pi}\left\{\sum_{k=0}^{\infty}\gamma^k r_{r+k+2} \mid s_{t+1} = s'\right\}\right]$$

$$= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}\left[R^a_{ss'} + \gamma V^{\pi}(s')\right]$$

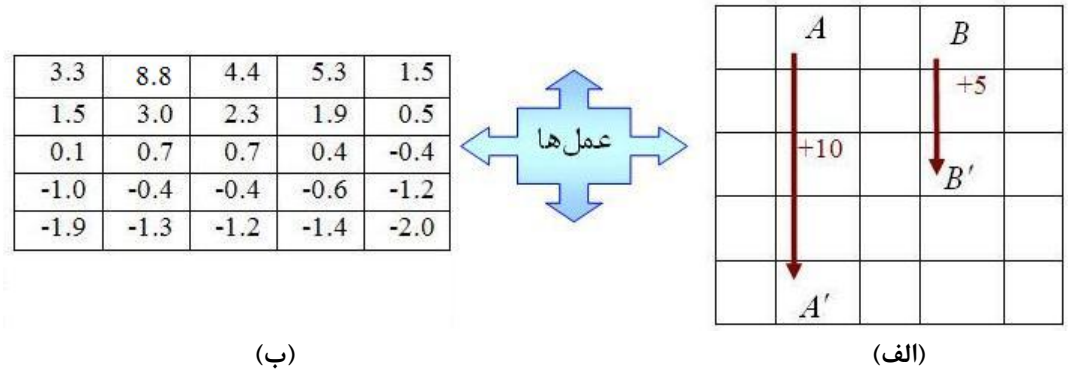
معادله (۹-۶) به معادله بلمن[۴] برای V^{π} معروف است. این معادله رابطه بین ارزش حالت s و ارزش حالات وقوع یافته‌ی بلافاصله بعد از آن را بیان می‌کند. تابع ارزش V^{π} حل یکتای معادله بلمن (۹-۶) می‌باشد. واضح است که اگر n حالت وجود داشته باشد و نیز پویایی محیط یعنی R^{search} و R^{wait} معلوم باشد، معادله بلمن به صورت دستگاه n معادله n مجهول درخواهد آمد که V^{π} حل یکتای آن خواهد بود.

مثال (ث): جدول زیر را در نظر بگیرید. خانه‌های این جدول متناظر با یک حالت از محیط می‌باشند. در هر خانه چهار عمل بالا، پایین، راست و چپ متصور است که انتخاب هر کدام از این عمل‌ها با

---

۱ monte carlo
۲ dynamic programming
۳ successor states
۴ bellman equetion

احتمال یک منجر به حرکت در همان جهت می‌شود (به‌عبارت دیگر محیط دارای رفتار قطعی[۱] است نه تصادفی[۲]). عمل‌هایی که منجر به خروج از جدول می‌شوند، مکان عامل را تغییر نمی‌دهند (به طور مثال اگر عامل در یکی از خانه‌های پایانی سمت راست باشد و عمل راست را انتخاب کند، سر جای خود باقی می‌ماند) و دارای امتیاز ۱- خواهند بود. اگر عامل در خانه A باشد، هر یک از چهار عمل باعث می‌شود که به خانه A' رفته و ۱۰+ امتیاز بگیرد. به‌همین ترتیب اگرعامل در خانه B باشد، هر یک از چهار عمل باعث می‌شود که به خانه B' رفته و ۵+ امتیاز بگیرد. در هر یک از خانه‌های دیگر هر عمل منجر به حرکت در همان جهت می‌شود و دارای امتیاز صفر است. فرض کنید عامل سیاستی را در پیش می‌گیرد که در آن احتمال انتخاب هر یک از چهار عمل کاملا یکنواخت است (سیاست یکنواخت) شکل ۶-۳ مقادیر تابع ارزش را در هر یک از حالت‌ها (خانه‌های جدول) نشان می‌دهد.



شکل ۶-۳ : مقادیر تابع ارزش در هر یک از حالت‌ها [۵۸].

این مقادیر برای $\gamma = 0.9$ و با حل دستگاه معادلات (۶-۹) محاسبه شده است. اعداد منفی که در خانه‌های کنارِضلع پایین جدول مشاهده می‌شود، بیان‌گر این است که اگر در این خانه‌ها باشیم و سیاست یکنواخت در پیش گرفته شود به احتمال زیاد از محدوده جدول خارج شده و امتیاز منفی کسب می‌شود. ارزش خانه‌های کنار ضلع بالا منفی نیست چون احتمال دارد از آن‌جا به A یا B منتقل شود که در آن‌صورت امتیازهای مثبتی دریافت می‌کند. این امتیازهای مثبت اثر امتیاز منفی حاصل از خروج از جدول را خنثی می‌کند. از طرف دیگر خانه A بیشترین ارزش را دارد اما اگر دقت شود امید ریاضی امتیاز کسب شده در این حالت کمتر از امتیازی است که پس از منتقل شدن به خانه A' یعنی ۱۰+ به دست می‌آورد. علت این است که پس از منتقل شدن به خانه A' به احتمال از جدول خارج شده و امتیاز منفی کسب می‌کند.

---
۱ deterministic
۲ probabilistic

۶-۷ تابع ارزش بهینه[۱]

می‌توان گفت حل مساله یادگیری تقویتی یعنی به دست آوردن سیاستی که با عمل به آن بیشترین پاداش انتظاری در بلند مدت کسب می‌شود. سیاست π را بهتر از سیاست π' گوییم هرگاه مقدار تابع ارزش آن برای تمامی حالت‌ها از تابع ارزش π' بیشتر باشد. به‌عبارت دیگر داشته باشیم:

$$V^{\pi}(s) \geq V^{\pi'}(s) \quad for \quad all \quad s \in S \qquad (۱۰-۶)$$

سیاست بهینه را سیاستی گوییم که از سایر سیاست‌ها بهتر باشد و تابع ارزش مربوط به آن تابع ارزش حالت بهینه گفته می‌شود که به صورت زیر تعریف می‌شود:

$$V^{*}(s) = \max_{\pi} V^{\pi}(s) \quad for \quad all \quad s \in S \qquad (۱۱-۶)$$

به همین ترتیب تابع ارزش عمل بهینه به صورت زیر تعریف می‌شود:

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a) \quad for \quad all \quad s \in S \quad and \quad a \in A \qquad (۱۲-۶)$$

برای هر جفت حالت و عمل (s,a)، این تابع مقدار پاداش انتظاری کلی را می‌دهد که اگر در حالت s عمل a انتخاب شود و سپس سیاست بهینه در پیش گرفته شود. با توجه به این واقعیت، می‌توان رابطه‌ای بین این دو تابع بهینه نوشت:

$$Q^{*}(s,a) = E\{r_{t+1} + \gamma V^{*}(s_{t+1}) \mid s_t = s, a_t = a\} \qquad (۱۳-۶)$$

چون V^{*} خود یک تابع ارزش حالت برای سیاست بهینه است، باید در معادله بلمن نیز صدق کند. ازطرفی چون این تابع ارزش، بهینه است می‌توان معادله بلمن را به صورت زیر که به معادله بهینگی بلمن[۲] معروف است، نوشت:

$$V^{*}(s) = \max_{a \in A} Q^{\pi^{*}}(s,a) \qquad (۱۴-۶)$$

$$= \max_{a} E_{\pi^{*}}\{R_t \mid s_t = s, a_t = a\}$$

$$= \max_{a} E_{\pi^{*}}\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \mid s_t = s, a_t = a\}$$

$$= \max_{a} E_{\pi^{*}}\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^k r_{t+k+2} \mid s_t = s, a_t = a\}$$

---

۱ optimal value function
۲ bellman optimality equation

$$= \max_a E\{r_{t+1} + \gamma V^*(s_{t+1}) \mid s_t = s, a_t = a\}$$

$$= \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')]$$

به همین ترتیب معادله بهینگی بلمن برای Q^* به صورت زیر است:

$$Q^*(s,a) = E\{r_{t+1} + \gamma \max_{a'} Q^*(s_{t+1}, a') \mid s_t = s, a_t = a\} \qquad (۱۵\text{-}۶)$$

$$= \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \max_{a'} Q^*(s', a')]$$

برای مسایل MDP متناهی، معادله (۶-۱۴) دارای جواب یکتا می‌باشد. در واقع معادله بهینگی بلمن یک دستگاه معادلات است که به اندازه تعداد حالات، متغیر دارد. حال هنگامی‌که V^* از طریق حل دستگاه یا هر روش دیگر به دست بیاید، از روی آن به‌سادگی می‌توان سیاست بهینه را استخراج کرد. برای هر حالت s یک یا چند عمل (به عنوان مثال a_1, a_2, a_3) وجود دارد که معادله بهینگی بلمن را بیشینه می‌کند. حال هر سیاست $\pi(s,a)$ که به اعمال a_1, a_2, a_3 احتمالِ انتخاب شدن ناصفر و به بقیه اعمال، احتمال انتخاب شدن صفر اختصاص بدهد، بهینه است. حال اگر به جای V^*، Q^* را داشته باشیم، به دست آوردن سیاست بهینه آسان‌تر نیز می‌شود. چون سیاست بهینه سیاستی خواهد بود که در حالت s به عمل‌هایی که معادله بهینگی بلمن برای Q^* را بیشینه می‌کنند، احتمال انتخاب شدن ناصفر و به بقیه اعمال احتمال انتخاب شدن صفر اختصاص بدهد. بنابراین در ازای تلاش بیشتر برای به دست آوردن $Q(s,a)$ به جای $V(s)$، Q^* به ما اجازه انتخاب اعمال بهینه را می‌دهد بدون این که نیازی باشد تا حالات بلافاصله وقوع یافته[۱] بعد از هر حالت را دانست و به طور کلی بدون این که نیاز باشد تا چیزی درباره دینامیک محیط دانست.

مثال (ج): معادله بهینگی بلمن برای روبات قوطی جمع کن: با استفاده از معادله (۶-۱۴) می‌توان V^* را برای روبات مثال (ت) به دست آورد. برای راحتی نمادهای l و h را برای حالات low و high و نمادهای s,w,r برای عمل‌های search, wait, recharge به کار برده می‌شود. چون در این مثال تنها دو حالت وجود دارد، بنابراین برای به دست آوردن تابع ارزش بهینه به یک دستگاه دو معادله دو مجهول مانند زیر نیاز است:

$$V^*(h) = \max \begin{Bmatrix} P^s_{hh}[R^s_{hh} + \gamma V^*(h)] + P^s_{hl}[R^s_{hl} + \gamma V^*(l)], \\ P^w_{hh}[R^w_{hh} + \gamma V^*(h)] + P^w_{hl}[R^w_{hl} + \gamma V^*(l)] \end{Bmatrix} \qquad (۱۶\text{-}۶)$$

$$= \max \begin{Bmatrix} \alpha[R^s + \gamma V^*(h)] + (1-\alpha)[R^s + \gamma V^*(l)], \\ 1[R^w + \gamma V^*(h)] + 0[R^w + \gamma V^*(l)] \end{Bmatrix}$$

[۱] successor states

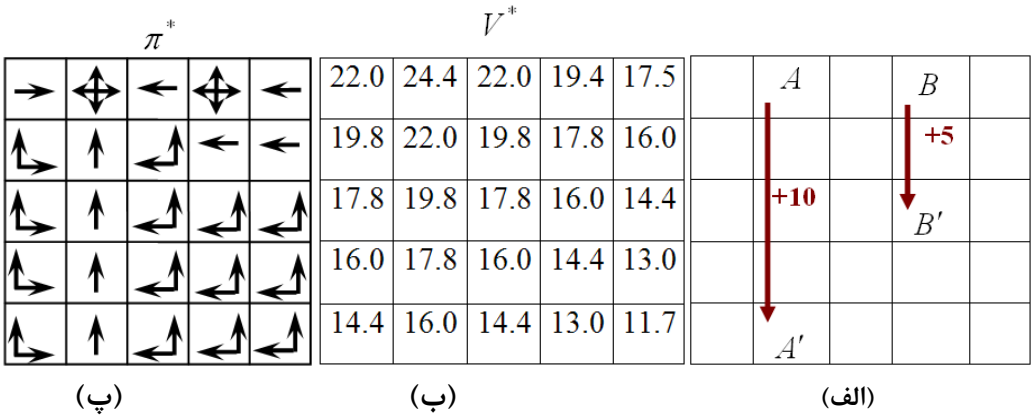
$$= \max\begin{Bmatrix} R^s + \gamma[\alpha V^*(h) + (1-\alpha)V^*(l)], \\ R^w + \gamma V^*(h) \end{Bmatrix}$$

اگر همین مراحل برای $V^*(l)$ طی شود تساوی زیر حاصل می‌شود:

$$V^*(l) = \max\begin{Bmatrix} \beta R^s - 3(1-\beta) + \gamma(1-\beta)V^*(h) + \beta V^*(l)], \\ R^w + \gamma V^*(l), \\ \gamma V^*(h) \end{Bmatrix} \qquad (۱۷\text{-}۶)$$

برای هر مقدار $0 \le \gamma \prec 1,\ 0 \le \alpha, \beta \le 1,\ R^w, R^s$ دستگاه غیر خطی بالا یک مقدار یکتا برای $V^*(h), V^*(l)$ می‌دهد.

مثال (د): مثال (ث) را در نظر بگیرید. فرض کنید که معادله بهینگی بلمن برای V^*، برای این مساله حل شده باشد. شکل ۶-۴ به ترتیب صورت مساله V^* و π^* را نشان می‌دهد:



شکل ۶-۴ : صورت مساله V^* و π^* [۵۸].

خانه‌ای که چند فلش دارد نشان دهنده این است که هر کدام از عمل‌ها بهینه است. حل کردن معادله بهینگی بلمن برای به دست آوردن جواب مساله یادگیری تقویتی به ندرت به طور مستقیم استفاده می‌شود. این حل بر سه فرض استوار است که این فرض‌ها به ندرت در دنیای واقعی برقرار است: (۱) ما به طور دقیق پویایی محیط را می‌دانیم. (۲) ما توانایی محاسبات برای حل کردن معادله را داریم. (۳) خاصیت مارکوف برقرار است. برای حل مساله‌ای که در دنیای بیرون با آن مواجه هستیم، به طور معمول چند تا از فروض فوق برقرار نیست به عنوان مثال ممکن است در مساله ای با 10^{20} حالت مواجه باشیم که حل کردن معادله بلمن برای آن هزاران سال طول می‌کشد. به خاطر همین موضوع در خیلی از مسایل یادگیری تقویتی از حل تقریبی معادله بهینگی بلمن استفاده می‌شود. در حل تقریبی، از مقادیر تجربی و کمیت‌هایی که به وسیله گذر از یک حالت به حالت دیگر و دریافت پاداش به وسیله یک عمل خاص به دست می‌آید به جای مقادیر دقیق و انتظاری ریاضی استفاده می‌شود.

۶-۸ برنامه‌نویسی پویا

برنامه‌نویسی پویا یا DP[۱] به مجموعه الگوریتم‌هایی اطلاق می‌شود که با در دست داشتن مدل کاملی ازمحیط به‌عنوان یک فرآیند تصمیم‌گیری مارکوف (MDP) می‌توانند سیاست بهینه[۲] را محاسبه کنند. الگوریتم‌های سنتی برنامه‌نویسی پویا گرچه از لحاظ تئوری قابل توجه‌اند ولی به خاطر احتیاج به مدل کاملی از محیط و هم‌چنین حجم محاسبات گسترده، از اهمیت کمی در یادگیری تقویتی برخوردارند.

برای ادامه دادن به بحث فرض می‌شود که با یک فرآیند مارکوف متناهی روبرو هستیم. به عبارت دیگر فرض می‌کنیم مجموعه حالت‌ها و مجموعه عمل‌هایی که در هر حالت امکان پذیر است، $A(s)$, s متناهی است و پویایی محیط به وسیله مجموعه احتمالات گذر $P^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$ و پاداش‌های انتظاری $R^a_{ss'} = E\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\}$ مشخص شده است. ایده کلیدیِ برنامه‌نویسی پویا، استفاده از توابع ارزش برای جستجو به منظور یافتن سیاست‌های مطلوب می‌باشد.

۶-۹ ارزیابی سیاست[۳]

در ابتدا بررسی می‌شود که چگونه می‌توان تابع ارزش حالت V^π را برای یک سیاست دلخواه مانند π به دست آورد. به این‌کار ارزیابی سیاست یا مساله تخمین[۴] گفته می‌شود. برای یاد آوری معادله بلمن برای V^π دوباره نوشته می‌شود:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V^\pi(s')] \qquad (۱۸-۶)$$

یک روش برای تخمین تابع ارزش حالت به این صورت می‌باشد: مجموعه حالت‌ها S و دنباله توابع $V_0, V_1, V_2, \ldots$ را در نظر بگیرید. هر کدام از این توابع نگاشتی از مجموعه حالت‌ها یعنی S به مجموعه اعداد حقیقی یعنی R می‌باشد. V_0 را تابعی دلخواه بگیرید. بقیه توابع $V_1, V_2, \ldots$ با توجه به رابطه بازگشتی زیر ساخته می‌شوند:

$$V_{k+1}(s) = E_\pi\{r_{t+1} + \gamma V_k(s_{t+1}) \mid s_t = s\} \qquad (۱۹-۶)$$

$$= \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} [R^a_{ss'} + \gamma V_k(s')$$

---

[۱] Dynamic Programming
[۲] optimal policy
[۳] policy evaliuation
[۴] prediction problem

می‌توان نشان داد با برقراری شرایطی که تحت آن وجود V^{π} ثابت می‌شود، هم‌گرایی دنباله فوق نیز برقراراست. پس اگر قرار باشد که دنباله فوق هم‌گرا شود با توجه به این که طبق معادله بلمن V^{π} یک حل برای حد این دنباله است و نیز مقدار V^{π} یکتاست، می‌توان نتیجه گرفت که دنباله توابع فوق به V^{π} هم‌گرا خواهد شد. به این الگوریتم، ارزیابی بازگشتی سیاست[۱] گویند.

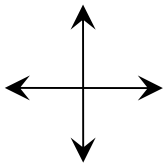
برای نوشتن کد الگوریتم فوق نیاز به استفاده از دو آرایه می‌باشد: یکی برای نگه‌داری مقادیرقدیمی یعنی V_k و دیگری برای نگه‌داری مقادیر جدید یعنی V_{k+1} به این صورت می‌توان مقادیر جدید تابع حالت را یکی یکی از مقادیر قدیمی آن به دست آورد بدون این که مقادیر قدیمی از بین برود. البته ساده‌تر خواهد بود که از یک آرایه برای نگه‌داری مقادیر استفاده شود و با به‌دست آوردن مقدار جدید آن جایگزین مقدار قدیمی شود. در این حالت بسته به ترتیبی که مقدار ارزش حالت‌ها به روز می‌شوند، بعضی مواقع به جای مقادیر قدیمی از مقادیر جدید در سمت راست معادله (۶-۱۹) استفاده می‌شود. ثابت می‌شود این الگوریتم برای ارزیابی سیاست که با الگوریتم قبلی اندکی متفاوت است نیز به V^{π} هم‌گرا خواهد شد. در واقع سرعت هم‌گرایی این الگوریتم که in-place نامیده می‌شود، بیشتر از الگوریتم قبلی است چون به محض ساخته شدن داده‌ها در به روز کردن مقادیر از آن‌ها استفاده می‌شود. نکته مهم دیگری که وجود دارد درباره نقطه پایان الگوریتم است. یک پیشنهاد برای پایان دادن به الگوریتم به این صورت است که در هر مرحله که مقادیر جدید از مقادیر قدیمی ساخته می‌شوند، مقدار $\max_{s\in S}|V_{k+1}(s)-V_k(s)|$ به دست آورده شود. هرگاه این مقدار از یک مقدار فرضی کوچک مانند ε کمتر شد می‌توان به الگوریتم خاتمه داد پس با توجه به توضیحات بالا می‌توان کد الگوریتم را به صورت شکل۶-۵ نوشت:

1. **Begin**
2. input π, the policy to be evaluated
3. initialize V(s)=0 for all $s \in S$
4. do{
5. $\Delta \leftarrow 0$
6. while ($s \in S$){
7. $v \leftarrow V(s)$
8. $V(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V\pi(s')]$
9. $\Delta \leftarrow \max(\Delta, |v - V|)$ }//end of while
10. } while ($\Delta \le \theta$ (a small positive number))//end of do
11. output $V \approx V^{\pi}$
12. **End**

شکل ۶-۵ : صورت مساله V^* و π^* [۵۸].

[۱] iterative policy evaluation

مثال (۵): مربع 4×4 زیر را در نظر بگیرید:



عمل‌ها

	1	2	3
4	5	6	7
8	9	10	11
12	13	14	

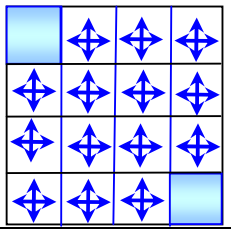
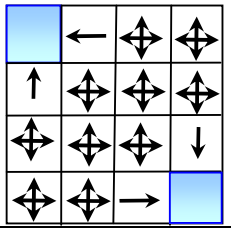
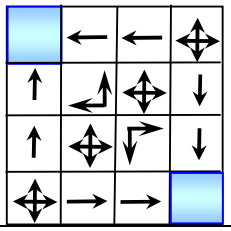
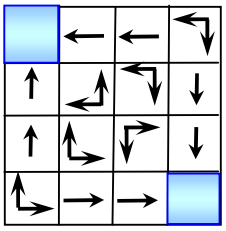
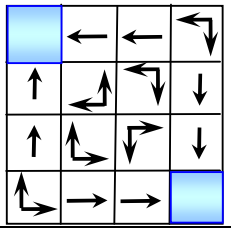
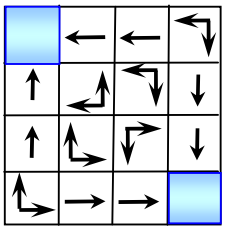
برای تمام حرکات
$r = -1$

شکل ۶-۶

فضای حالت به صورت $S = \{1, 2, 3, ..., 14\}$ و اعمال ممکن در هر حالت به صورت $A = \{up, down, right, left\}$ است. انجام هر عمل با احتمال یک منجر به حرکت در همان جهت خواهد شد تنها اعمالی که منجر به خروج از جدول می‌شوند که در اثر این اعمال، عامل در همان مکان قبلی خود باقی می‌ماند. پس به عنوان مثال داریم:

$$P_{5,6}^{right} = 1,\ P_{5,10}^{right} = 0,\ P_{7,7}^{right} = 1$$

دو خانه‌ای که در جدول پر شده‌اند، خانه‌های مطلوب در نظر گرفته می‌شوند. هر حرکتی که منجر به انتقال به یکی از خانه‌های مطلوب شود دارای پاداش صفر و در غیر این صورت دارای پاداش ۱- خواهد بود. فرض کنید عامل از سیاستی پیروی می‌کند که در آن احتمال حرکت در هر چهار جهت مساوی و برابر $\frac{1}{4}$ است. شکل ۶-۷ دنباله توابعِ ارزش حالت که به‌وسیله الگوریتم ارزیابی بازگشتی سیاست محاسبه شده‌اند را نشان می‌دهد.

	V_k: سیاست تصادفی	سیاست حریصانه	
K=0	0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0		سیاست تصادفی ←
K=1	0.0 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 -0.1 0.0		
K=2	0.0 -1.7 -2.0 -2.0 -1.7 -2.0 -2.0 -2.0 -2.0 -2.0 -2.0 -1.7 -2.0 -2.0 -1.7 0.0		
K=3	0.0 -2.4 -2.9 -3.0 -2.4 -2.9 -3.0 -2.9 -2.9 -3.0 -2.9 -2.4 -3.0 -2.9 -2.4 0.0		سیاست بهینه ←
K=10	0.0 -6.1 -8.4 -9.0 -6.1 -7.7 -8.4 -8.4 -8.4 -8.4 -7.7 -6.1 -9.0 -8.4 -6.1 0.0		سیاست بهینه ←
K=∞	0.0 -14.0 -20.0 -22.0 -14.0 -18.0 -20.0 -20.0 -20.0 -20.0 -18.0 -14.0 -22.0 -20.0 -14.0 0.0		سیاست بهینه ←

شکل ۶-۷ : دنباله توابعِ ارزش حالت [۵۸].

۶-۱۰ بهبود سیاست[۱]

در قسمت پیش به این خاطر سعی شد تا تابع ارزش یک سیاست پیدا شود چون هدف این بود که به وسیله آن سیاست‌های خوب را شناسایی کنیم. حال فرض کنید تابع ارزش برای یک سیاست قطعی[۲]، π به دست بیاید. سیاست قطعی سیاستی است که در هر حالت تنها یک عمل را پیشنهاد می‌کند و احتمال انتخابِ بقیه اعمال صفراست به‌عبارتی: $\pi(s,a)=1, \pi(s,b)=0 \ \forall b \neq a$. حال سوال این است که در حالت s آیا بهتر است که سیاست عوض شده و به جای انتخاب a عملی دیگر انتخاب شود یا در همان عمل قبلی باقی بماند؟ پاداش انتظاری حاصل از پیروی کردن از سیاست π با شروع از حالت s -که همان $V^{\pi}(s)$ است- مشخص است اما اگر سیاست عوض شود، آیا پاداش بیشتری حاصل نمی‌شود؟ یک راه برای پاسخ به این سوال این است که فرض شود در حالت s عمل a انجام داده شود و سپس از سیاست π پیروی شود. پاداش انتظاری حاصل از این عمل عبارت است از:

$$Q^{\pi}(s,a)=E_{\pi}\{r_{t+1}+\gamma V^{\pi}(s_{t+1})\,|\,s_t=s, a_t=a\}=\sum_{s'}P^{a}_{ss'}[R^{a}_{ss'}+\gamma V^{\pi}(s')] \quad (۲۰\text{-}۶)$$

نکته کلیدی که باید مشخص شود این است که آیا مقدار بالا از $V^{\pi}(s)$ بزرگ‌تر است یا نه. اگر مقدار بالا بزرگ‌تر باشد می‌توان نتیجه گرفت که هرگاه حالت s برقرار شود عمل a انتخاب شود و سپس از سیاست π پیروی شود بهتر است از این که پس از رسیدن به حالت s از سیاست π پیروی شود. پس در نتیجه سیاست جدید بهتر از سیاست π خواهد بود.

حالت بالا، یک حالت خاص از قضیه بهبود سیاست[۳] است. این قضیه می‌گوید: فرض کنید π, π' دو سیاست قطعی باشند به گونه‌ای که برای هر $s \in S$ وجود داشته باشد $Q^{\pi}(s,\pi'(s)) \geq V^{\pi}(s)$ آنگاه سیاست π' به همان خوبی سیاست π یا بهتر از آن است، یا به عبارت دیگر می‌توان نتیجه گرفت: $V^{\pi'}(s) \geq V^{\pi}(s)$.

ایده اثبات این قضیه به شکل زیر است:

$$V^{\pi}(s) \leq Q^{\pi}(s,\pi'(s))=E_{\pi'}\{r_{t+1}+\gamma V^{\pi}(s_{t+1})\,|\,s_t=s\} \quad (۲۱\text{-}۶)$$

$$\leq E_{\pi'}\{r_{t+1}+\gamma Q^{\pi}(s_{t+1},\pi'(s_{t+1}))\,|\,s_t=s\}$$

$$=E_{\pi'}\{r_{t+1}+\gamma E_{\pi'}\{r_{t+2}+\gamma V^{\pi}(s_{t+2})\}\,|\,s_t=s\}$$

$$=E_{\pi'}\{r_{t+1}+\gamma r_{t+2}+\gamma^2 V^{\pi}(s_{t+2})\,|\,s_t=s\}$$

---

۱ policy improvement
۲ deterministic
۳ policy improvement theorem

$$\leq E_{\pi'}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 V^{\pi}(s_{t+3}) \mid s_t = s\}$$

.

.

.

$$\leq E_{\pi'}\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \ldots \mid s_t = s\} = V^{\pi'}(s)$$

حال که بهبود یک سیاست به وسیله تغییر در انتخابِ عمل در یک حالت s مشاهده شد، طبیعی است که این ایده به تمام حالت‌ها و به تمام عمل‌های ممکن گسترش داده شود به این صورت که در هر حالت عملی انتخاب می‌شود که عبارت $Q^{\pi}(s, a)$ را بیشینه کند. پس برای ساختن سیاست بهترِ π' از روی سیاست π به این ترتیب عمل شود:

$$\pi'(s) = \arg\max_a Q^{\pi}(s, a) \qquad (۲۲-۶)$$

$$= \arg\max_a E\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s, a_t = a\}$$

$$= \arg\max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi}(s')]$$

جمله $\arg\max_a$ به معنی انتخاب آن مقدار از a است که عبارت بعد از $\arg\max_a$ بیشینه می‌کند. با توجه به روند ساخت π' مشاهده می‌شود که شرط قضیه بهبود سیاست رعایت شده است. بنابراین می‌توان گفت: $\pi'(s) \geq \pi(s)$

حال فرض کنید π' به همان خوبیِ π است (و نه بهتر از آن) یعنی $V^{\pi} = V^{\pi'}$. حال از معادله (۲۲-۶) خواهیم داشت:

$$V^{\pi'}(s) = \max_a E\{r_{t+1} + \gamma V^{\pi'}(s_{t+1}) \mid s_t = s, a_t = a\} \qquad (۲۳-۶)$$

$$= \max_a \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^{\pi'}(s')]$$

اما تساوی بالا همان معادله بهینگی بلمن است. پس هر دو سیاست‌های π' و π بهینه‌اند. بنابراین نتیجه می‌گیریم که روند بهبودسازی یک سیاست، سیاستی همیشه بهتر از سیاست قبلی را می‌دهد، مگر این که سیاست پیشین از قبل بهینه بوده باشد.

تمام ایده‌هایی که در بالا برای بهبود دادن یک سیاست قطعی ذکر شد، قابل تعمیم به سیاست‌های احتمالی[۱] می‌باشد. ستون دوم از شکل ۶-۶ نمونه‌ای از بهبود سیاست برای یک سیاست به طور کامل تصادفی را نشان می‌دهد. سیاست بهینه π^* در آخر به دست آمده است (در جداولی که در یک خانه چند جهت نشان داده شده است، نشان دهنده این است که هر کدام از این اعمال به هر احتمالی می‌توانند انتخاب شوند).

۶-۱۱ زنجیره تکامل سیاست[۲]

هنگامی‌که سیاست π مشابه آن‌چه در بخش قبل گفته شد و با استفاده از V^{π} به سیاست π' بهبود داده شود، می‌توان با محاسبه $V^{\pi'}$ و با روش مشابه سیاست π' را به سیاست π'' بهبود داد. به همین ترتیب می‌توان سیاست‌های حاصله را یکی یکی ارزیابی کرده و تابع ارزش حالتِ آن را به دست آورد و سپس آن را بهبود بخشید.

شکل ۶-۸ گویای روند عملکرد فوق است:

$$\pi_0 \xrightarrow{E} V^{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} V^{\pi_1} \xrightarrow{I} \pi_2 \xrightarrow{E} \dots \xrightarrow{I} \pi^* \xrightarrow{E} V^*$$

شکل ۶-۸ : زنجیره تکامل سیاست [۵۸].

در شکل فوق $\xrightarrow{E}$ نشان دهنده ارزیابی سیاست و $\xrightarrow{I}$ نشان دهنده بهبود سیاست است. هر سیاست از سیاست قبلی همیشه بهتر است (مگر این که سیاست قبلی بهینه بوده باشد) و چون مساله ما از نوع متناهی است و تعداد حالات متناهی می‌باشد، سیاست بهینه و تابع ارزش مربوط به آن طبق روند بالا در متناهی گام به دست می‌آید. به این روش پیدا کردن سیاست بهینه *Policy Iteration* گویند. شبه کد یافتن سیاست بهینه را می‌توان به صورت شکل ۶-۹ نوشت:

---

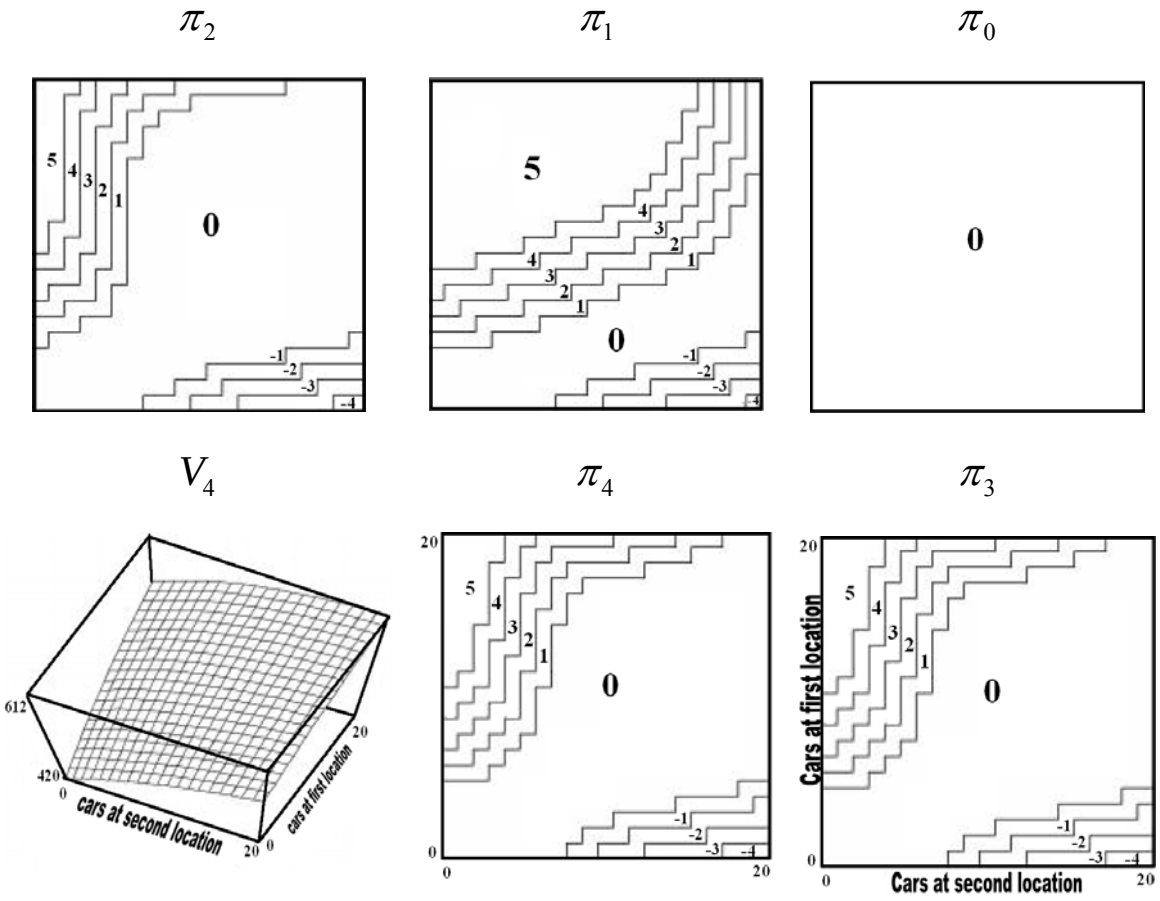
[۱] probabilistic
[۲] policy iteration

1. **Begin**
2. initialization
3. $V(s) \in R$ and $\pi(s) \in A(s)$ arbitrarily for all $s \in S$
4. policy evaluation
5. do{
6. $\Delta \leftarrow 0$
7. while($s \in S$)
8. $v \leftarrow V(s)$
9. $V(s) = \sum_{s'} P_{ss'}^{\pi(s)} \left[R_{ss'}^{\pi(s)} + \gamma V(s') \right]$
10. $\Delta \leftarrow \max(\Delta, |v - V(s)|)$ }//end of while
11. }while($\Delta \le \theta$ a small positive number)//end of do
12. policy improvement
13. policy-stable $\leftarrow$ true
14. while($s \in S$)
15. $b \leftarrow \pi(s)$
16. $\pi(s) = \arg\max_a \sum_{s'} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma V(s') \right]$
17. if ($b \neq \pi(s)$) $policy - stable \leftarrow false$ }//end of while
18. if (policy-stable) stop
19. else goto 2
20. **End**

شکل ۶-۹ : شبه کد یافتن سیاست بهینه [۵۸].

مثال (و): جک مدیر دو موسسه کرایه اتومبیل است. هر روز تعدادی مشتری به هر کدام از این موسسه‌ها برای کرایه اتومبیل مراجعه می‌کنند. اگر در آن موسسه اتومبیلی موجود باشد، جک آن را کرایه می‌دهد و ۱۰ دلار می‌گیرد. همچنین اتومبیل‌هایی که اجاره گرفته می‌شوند، روز بعد پس داده می‌شوند. جک برای این که مطمئن شود هنگام مراجعه مشتری، اتومبیل در موسسه موجود است می‌تواند شبانه اتومبیل‌ها را بین دو موسسه جابه‌جا کند که این کار برای او دو دلار هزینه دارد. فرض کنید تعداد اتومبیل‌هایی که در هر روز اجاره و پس داده می‌شوند برای هرکدام از موسسه‌ها یک متغیر تصادفی پواسن با میانگین λ باشد. پس داریم: $P(X = n) = \frac{e^{-\lambda}\lambda^n}{n!}$. همچنین فرض کنید که λ برای موسسه اول و دوم برای کرایه کردن به ترتیب برابر ۴ و ۳ و برای پس آوردن ۲ و ۳ است. برای سادگی فرض کنید که هر موسسه تنها گنجایش ۲۰ اتومبیل را دارد (اتومبیل‌های اضافی به نمایندگی مرکزی انتقال داده می‌شوند) و نیز حداکثر تعداد اتومبیل‌ها که شبانه می‌تواند بین دو موسسه جابه‌جا شود ۵ است. این مساله در قالب یک MDP متناهی با نرخ تنزیل $\gamma = 0.9$ فرمول‌بندی می‌شود. در اینجا گام‌های زمانی همان روزها، متغیر حالت تعداد اتومبیل‌های موجود در هر موسسه و عمل‌های ممکن تعداد خالص اتومبیل‌هایی است که شبانه بین دو موسسه جابه‌جا می‌شود. شکل زیر سیاست‌های به دست آمده از الگوریتم policy iteration را نشان می‌دهد. سیاست اولیه یا π_0 عبارت است از این که در هر حالتی که باشد هیچ اتومبیلی بین دو موسسه جابه‌جا نمی‌شود. ۵ نمودار اول به ازای هر تعداد اتومبیلی که در هر موسسه در پایان روز موجود باشد، تعداد اتومبیل‌هایی را نشان می‌دهد که باید شبانه

از موسسه اول به موسسه دوم انتقال یابد. بدیهی است اعداد منفی نشانگر جابه‌جایی از موسسه دوم به موسسه اول است. در نمودارهای زیر، هر سیاست از سیاست قبلی همیشه بهتر است و سیاست آخر بهینه می‌باشد.



شکل ۶-۱۰: محل قرار گیری اتومبیل‌ها [۵۸].

۶-۱۲ کارآیی برنامه‌نویسی پویا

گرچه ممکن است برنامه‌نویسی پویا برای حل مسایل بزرگ عملی نباشد اما برای حل مسایل MDP در مقایسه با سایر روش‌ها کار آمدتر است. اگر کمی از جزییات فنی صرف‌نظر شود، زمانی که طول می‌کشد تا برنامه‌نویسی پویا سیاست بهینه را پیدا کند یک چندجمله‌ای است که به تعداد حالات و عمل‌های ممکن بستگی دارد. یعنی اگر n حالت و m عمل ممکن داشته باشیم، زمان حل برای یافتن سیاست بهینه کمتر از یک چندجمله‌ای است که این چندجمله‌ای تابعی از m و n است. برنامه‌نویسی پویا سیاست بهینه را از بین m^n سیاست (قطعی) پیدا می‌کند که از هر الگوریتمِ جستجوی دیگری در فضای سیاست‌ها به‌صورت نمایی سریع‌تر است. برنامه‌نویسی خطی نیز برای حل

مسایل MDP کاربرد دارد و حتی در بعضی موارد سریع‌تر از برنامه‌نویسی پویا جواب می‌دهد اما تعداد حالت‌های[۱] مسایلی که می‌تواند حل کند بسیار کمتر از تعداد حالت‌های مسایل حل شده به وسیله برنامه‌نویسی پویا است (درمرتبه حدود ۱۰۰) بنابراین در مسایل گسترده[۲] تنها برنامه‌نویسی پویا عملی است. امروزه و در عمل کامپیوترها قادرند به روش برنامه‌نویسی پویا مسایل MDP با میلیون‌ها حالت را حل کنند.

۶-۱۳ خلاصه فصل

در این فصل در ابتدا مفهوم یادگیری تقویتی معرفی شده و سپس مفاهیم عامل، محیط، اهداف و پاداش که از اجزای یادگیری تقویتی می‌باشند شرح داده شدند. پس از آن فرآیند تصمیم‌گیری مارکوف و ارتباط آن با یادگیری تقویتی توضیح داده شد. بعد از آن مفهوم برنامه‌نویسی پویا به همراه یک مثال شرح داده شده و در نهایت کارآیی برنامه‌نویسی پویا خاتمه دهنده مطالب این فصل می‌باشد. در ادامه در فصل آتی خودکارهای یادگیر معرفی خواهد شد.

---

۱ state
۲ large problems

فصل هفتم

خودکارهای یادگیر

۷-۱ مقدمه

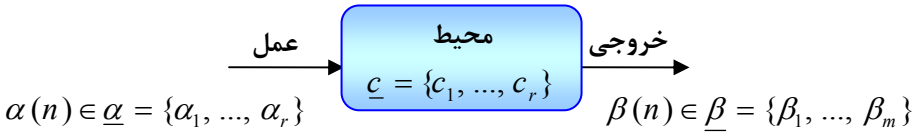
خودکار یادگیر (LA[۱]) یکی از شاخه‌های یادگیری ماشین[۲] به شمار می‌رود. روش خودکار یادگیر در یادگیری عبارت است از تعیین عمل بهینه از مجموعه‌ای محدود از اعمال از پیش تعریف شده که قابل انجام در یک محیط تصادفی[۳] می‌باشد. فرض می‌شود که خودکار در یک محیط ناشناس فعالیت می‌کند. در هر لحظه، عملی را از مجموعه اعمال خود انتخاب و به محیط اعمال می‌نماید. محیط در پاسخ به عمل انجام شده، یک خروجی از مجموعه خروجی‌های تعریف شده که می‌تواند محدود یا نا محدود باشد تولید و به خودکار اعلام می‌کند و خودکار با دریافت پاسخ محیط، شیوه تصمیم‌گیری خود را در انتخاب عمل بعدی به‌هنگام[۴] می‌کند. فرض می‌شود که بین هر عمل خودکار و پاسخ محیط یک رابطه احتمالی وجود داشته باشد و این رابطه که در واقع مشخصات داخلی محیط می‌باشد، در طول یادگیری به وسیله خودکار شناخته می‌شود. نظریه‌های خودکارهای یادگیر به تجزیه و تحلیل خودکارهایی که در چنین محیط‌هایی فعالیت می‌کنند می‌پردازد [۵۹].

---

۱ Learning Automata
۲ machine learning
۳ random environment
۴ update

۷-۲ محیط

کلمه محیط[۱] به مجموعه کلیه شرایط خارجی که زندگی و توسعه یک سازمان را تحت تاثیر قرار می‌دهد، اطلاق می‌گردد. پیاده‌سازی دقیق تعریف فوق به طور معمول یکی از مراحل بسیار مشکل حل مساله محسوب می‌گردد. در واقع این تعریف شامل شرایط بسیار زیادی می‌شودکه در طول حیات یک خودکار و یا گروهی از خودکارها تاثیر می‌گذارند. مدل ریاضی محیط به صورت سه تایی $\{\underline{\alpha}, \underline{c}, \underline{\beta}\}$ تعریف می‌شود که در آن $\underline{\alpha} = \{\alpha_1, ..., \alpha_r\}$ مجموعه محدود ورودی‌ها و $\underline{\beta} = \{\beta_1, ..., \beta_m\}$ یا $\underline{\beta} = \{(a, b) : a, b \in R\}$ مجموعه محدود یا نامحدود خروجی‌های محیط و $\underline{c} = \{c_1, ..., c_r\}$ مجموعه محدودی از احتمال‌های جریمه[۲] می‌باشد. هر c_i با α_i رابطه دارد و در واقع c_i مشخصات و رفتار محیط را تعریف می‌کند. شکل ۷-۱ شکل کلی مدل محیط را نمایش می‌دهد [۳].



شکل ۷-۱ : مدل محیط تصادفی [۲].

بر حسب نحوه تعریف مجموعه $\underline{\beta}$، مدل‌های مختلفی از محیط تعریف می‌گردد، اما اغلب پاسخ محیط به عمل ورودی به یکی از دو صورت مثبت یا منفی (عمل مطلوب یا نامطلوب) در نظر گرفته می‌شود که در این حالت مجموعه $\underline{\beta}$ به صورت یک مجموعه دو عضوی $\underline{\beta} = \{\beta_1, \beta_2\}$ تعریف می‌گردد و به طور معمول با دو علامت $\underline{\beta} = \{0, 1\}$ نمایش داده می‌شود. در خودکارهای یادگیر، به طور معمول 0 به عنوان پاسخ مطلوب و 1 به عنوان پاسخ نامطلوب در نظر گرفته می‌شود. ورودی $\alpha(n)$ در زمان‌های گسسته $t = n,\ n \in \{0, 1, ...\}$ به محیط اعلام می‌گردد و محیط در پاسخ، خروجی $\beta(n) \in \underline{\beta}$ را تولید می‌کند و با این خروجی عمل انجام شده را ارزیابی می‌کند. این خروجی‌ها با توجه به مشخصات داخلی (c_i) محیط و عمل انجام شده تولید می‌گردد.

$$c_i = \Pr\{\beta(k) = 1 \mid \alpha(k) = \alpha_i\} \qquad (۱-۷)$$

c_i احتمال دریافت پاسخ نامطلوب از محیط است، اگر عمل α_i در محیط انجام شود، طبق تعریف فوق محیط به صورت ارتباط بین هر عمل ورودی و پاسخ خروجی تعریف می‌گردد.

---

[۱] environment
[۲] penalty probabilities

در سامانه‌های پویا که اعمال به طور پیوسته در محیط انجام می‌شوند. مجموعه اعمال، مجموعه محدودی است که در یک بازه زمانی محدود در محیط انجام می‌شوند. با در نظرگرفتن یک مقدار آستانه[۱] برای تعریف پاسخ دودویی محیط، خروجی محیط در پایان هر بازه زمانی تعریف می‌گردد. در این حالت $\alpha(n)$ بیان‌گر عمل انجام گرفته در بازه (n - 1, n) و خروجی $\beta(n)$ محیط مقدار خروجی در لحظه n در نظر گرفته می‌شود. بر حسب این که مجموعه β چگونه تعریف می‌گردد، سه مدل مختلف برای محیط تعریف می‌گردد که عبارتند از:

مدل P: در این مدل خروجی محیط به صورت یک مجموعه دودویی که شامل پاسخ مطلوب یا نامطلوب می‌باشد، تعریف می‌گردد. این مدل، در مواردی ممکن است مجبور به استفاده از یک مقدار آستانه جهت تعیین مطلوب یا نامطلوب بودن پاسخ محیط شود.

مدل Q: در این مدل مجموعه خروجی محیط به صورت مجموعه گسسته $\underline{\beta} = \{\beta_1, ..., \beta_m\}, m > 2$ تعریف می‌گردد. در این مدل، خروجی محیط به صورت m مقدار گسسته که در بازه [0, 1] تعریف می‌گردد و به طور معمول مقدار بزرگ‌تر به عنوان پاسخ مطلوب‌تر در نظر گرفته می‌شود.

مدل S: در این مدل مجموعه خروجی محیط به صورت مجموعه پیوسته $\underline{\beta} = \{(a, b), 0 \le a, b \le 1\}$ در نظر گرفته می‌شود. در این مدل نیز به طور معمول مقدار بزرگ‌تر برای پاسخ محیط، به عنوان پاسخ مطلوب‌تردر نظر گرفته می‌شود.

مدل P برای بسیاری از کاربردها مناسب می‌باشد. اما مدل‌های S و Q انعطاف بیشتری به خودکارهای یادگیر می‌دهند. اگر مشخصات (c_iها) در طول زمان تغییر نکنند، محیط ایستا[۲] می‌باشد. اما در عمل یک یا چند c_i می‌توانند در طول زمان تغییرکنند به چنین محیط‌هایی، محیط غیر ایستا[۳] گفته می‌شود. بر حسب این که تغییرات مشخصات محیط چگونه می‌باشد انواع مختلفی از مدل‌های محیط تعریف می‌گردد که عبارتند از:

۱. محیط‌های دوره‌ای[۴]
۲. محیط‌های با تغییرات کند[۵]
۳. محیط‌های با تغییرات تصادفی[۶]

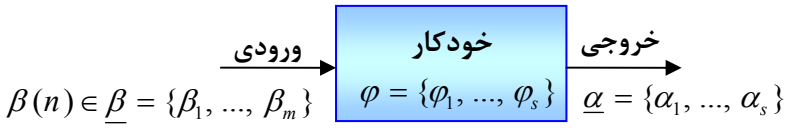
---

۱ threshold
۲ stationary
۳ non-stationary
۴ periodic
۵ slow verity
۶ random verity

در مدل یادگیری گفته شده، یادگیری شامل درک میزان تاثیر هر عمل ورودی بر محیط و یا فهم رابطه متقابل ورودی و خروجی محیط می‌باشد. اگر تمامی c_iها از قبل معلوم باشد، تاثیر هر عمل α_i بر محیط معلوم خواهد شد. در این صورت عمل α_i متناظر با c_l $(c_l = \min_i \{c_i\})$ بهترین عمل است. در عمل c_iها ناشناخته هستند. در این شرایط یادگیری به معنی تعیین تاثیر هر عمل روی محیط و به هنگام‌سازی استراتژی انتخاب عمل، جهت انتخاب عمل و تحلیل تاثیر آن بر محیط توسعه خودکار می‌باشد.

۷-۳ خودکار

خودکار، یک سامانه مجرد است که به صورت ۵تایی $\{\underline{\varphi}, \underline{\alpha}, \underline{\beta}, F(o,o), H(o,o)\}$ تعریف می‌شود که در آن $\underline{\varphi}$ مجموعه حالات داخلی، $\underline{\alpha}$ مجموعه خروجی، $\underline{\beta}$مجموعه ورودی، $F(o,o): \underline{\varphi} \times \underline{\beta} \to \underline{\varphi}$ تابع تغییر حالت بر حسب حالت و ورودی فعلی و $H(o,o): \underline{\varphi} \times \underline{\beta} \to \alpha$ تابع خروجی خودکار بر حسب حالت و ورودی فعلی می‌باشد. در صورتی که خروجی خودکار تنها به حالت فعلی آن بستگی داشته باشد، به آن خودکار حالت-خروجی[۱] گفته می‌شود. در این حالت، تابع H (o, o) به تابع $G(o): \underline{\varphi} \to \underline{\alpha}$ تبدیل می‌شود. شکل ۷-۲ شکل کلی از خودکار یادگیر را نشان می‌دهد [۳].



شکل ۷-۲ : مدل خودکار یادگیر [۲].

به طور رسمی مشخصات خودکار به صورت زیر تعریف می‌شود:

حالت خودکار در لحظه n: $\varphi(n) \in \underline{\varphi} = \{\phi_1, ..., \varphi_s\}$

خروجی یا عمل خودکار در لحظه n: $\alpha(n) \in \underline{\alpha} = \{\alpha_1, ..., \alpha_r\}$

ورودی خودکار در لحظه n که می‌تواند عضوی از مجموعه محدود یا نامحدود ورودی‌ها باشد:

$$\beta(n) \in \underline{\beta} = \{\beta_1, ..., \beta_m\} \quad \text{یا} \quad \beta(n) \in \underline{\beta} = \{(a,b)\} \quad , a, b \in R \qquad (۷-۲)$$

---

[۱] state action

تابع تبدیل حالت خودکار F (o, o) که حالت خودکار در لحظه n + 1 را با توجه به حالت و ورودی در لحظه n تعیین می‌کند:

$$\varphi(n+1) = F[\varphi(n), \beta(n)] \qquad (۳-۷)$$

تابع خروجی G(o) خودکار که خروجی در لحظه n را بر حسب حالت آن در لحظه n تعیین می‌کند.

$$\alpha(n) = G[\varphi(n)] \qquad (۴-۷)$$

تعاریف فوق برای خودکار حالت-خروجی بیان شده‌اند. می‌توان نشان داد که برای هر خودکار با تابع خروجی H (o, o) می‌توان خودکار معادل دیگری با تابع خروجی G(o) پیدا کرد. عملکرد خودکار در زمان‌های n = 0, 1, 2, … به صورت زیر می‌باشد:

با داشتن حالت اولیه $\varphi(0)$ عمل $\alpha(0)$ با توجه به تابع G (0) انتخاب و به محیط اعلام می‌گردد. با دریافت پاسخ $\beta(0)$ و با استفاده از تابع F (o, o) حالت خودکار در لحظه n = 1 یعنی $\varphi(1)$ تعیین می‌گردد و این چرخه ادامه پیدا می‌کند. مشاهده می‌شود که حالت و عمل خودکار در لحظه n فقط به ورودی و حالت خودکار در لحظه n - 1 بستگی دارد. به این ترتیب تناظری بین خودکارهای یادگیر و زنجیره‌های مارکوفی[۱] به وجود می‌آید که در تحلیل نحوه عملکرد خودکار می‌تواند مفید باشد.

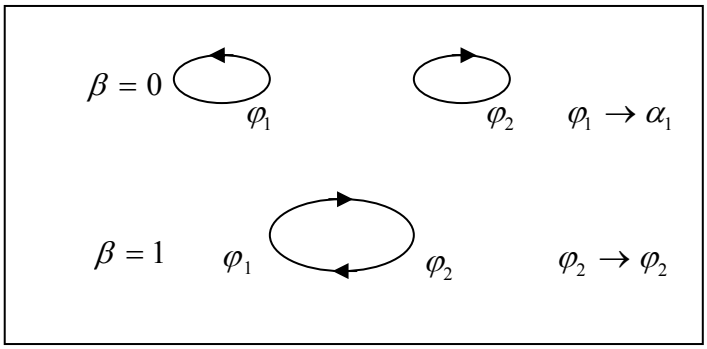
در صورتی که هر دو تابع F و G قطعی[۲] باشند، خودکار قطعی است، در این صورت با داشتن حالت و ورودی فعلی، خروجی فعلی و حالت بعدی خودکار به طور قطع قابل پیش بینی خواهد بود. اگر F یا G غیرقطعی باشند، خودکار غیر قطعی[۳] خواهد بود. در این صورت با داشتن حالت و ورودی فعلی خودکار، خروجی فعلی خودکار، خروجی فعلی و حالت بعدی خودکار را به طور قطع نمی‌توان تعیین کرد بلکه فقط احتمال رخداد هر یک را می‌توان تعیین نمود.

۷-۳-۱ خودکار قطعی

مثالی از یک خودکار قطعی با دو حالت φ_1 و φ_2 و دو عمل α_1 و α_2 با بلوک دیاگرام شکل ۷-۳ را در نظر بگیرید.

---

۱ markov-chain
۲ deterministic
۳ non-deterministic



شکل ۷-۳: شکل خودکار قطعی [۲].

در این خودکار با دریافت پاسخ مطلوب در حالت فعلی باقی می‌ماند و در صورت دریافت پاسخ نامطلوب تغییر حالت می‌دهد. نمایش گرافیکی فوق را می‌توان به فرم ماتریسی زیر نمایش داد. ماتریس $F(\beta)$، ماتریس تغییر حالت خودکار بر حسب ورودی β و ماتریس G ماتریس خروجی می‌باشند.

$$F(0) = \begin{matrix} \\ \varphi_1 \\ \varphi_2 \end{matrix} \begin{matrix} \begin{matrix} \varphi_1 & \varphi_2 \end{matrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{matrix} \quad ، F(1) = \begin{matrix} \\ \varphi_1 \\ \varphi_2 \end{matrix} \begin{matrix} \begin{matrix} \varphi_1 & \varphi_2 \end{matrix} \\ \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{matrix} \quad ، G = \begin{matrix} \\ \varphi_1 \\ \varphi_2 \end{matrix} \begin{matrix} \begin{matrix} \alpha_1 & \alpha_2 \end{matrix} \\ \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{matrix} \qquad (۵-۷)$$

به طور رسمی مولفه‌های ماتریس F و G به صورت زیر قابل بیان هستند:

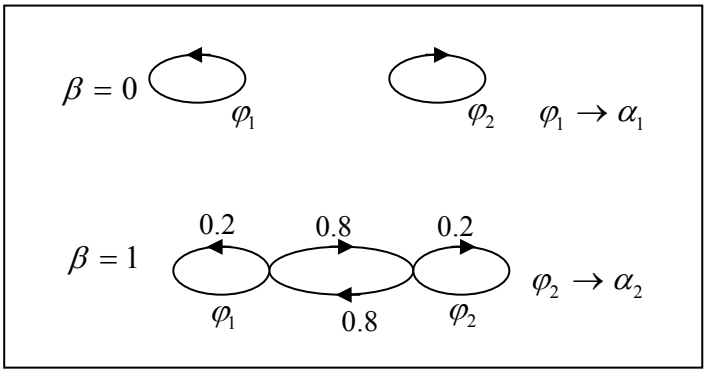
$$f_{ij}^{\beta} = \begin{cases} 1 & if \quad \varphi_i \to \varphi_j \; for \quad input \quad \beta \\ 0 & else \end{cases} \qquad (۵-۷)$$

$$g_{ij}^{\beta} = \begin{cases} 1 & if \quad G(\varphi_i) = \alpha_j \\ 0 & else \end{cases}$$

به طور کلی، ماتریس تبدیل حالت F یک خودکار قطعی را می‌توان به وسیله m ماتریس $s \times s$ بیان کرد به طوری که m تعداد ورودی‌های ممکن و s تعداد حالات باشد. به طور مشابه تابع خروجی G به وسیله یک ماتریس $s \times r$ که r تعداد خروجی‌ها است، قابل بیان می‌باشد. ماتریس‌های فوق ، ماتریس‌های دودویی هستند و در هر سطر تنها یک مقدار ۱ می‌تواند وجود داشته باشد.

۷-۳-۲ خودکار غیر قطعی

اگر F یا G احتمالی[۱] باشند، در این صورت خودکار غیر قطعی خواهد بود. مثال قبل را می‌توان به صورت شکل ۷-۴ اصلاح کرد.



شکل ۷-۴ : شکل خودکار غیرقطعی [۲].

$$F(0) = \begin{matrix} & \varphi_1 & \varphi_2 \\ \varphi_1 \\ \varphi_2 \end{matrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad ، \quad F(1) = \begin{matrix} & \varphi_1 & \varphi_2 \\ \varphi_1 \\ \varphi_2 \end{matrix} \begin{bmatrix} 0.2 & 0.8 \\ 0.8 & 0.2 \end{bmatrix} \quad ،G = \begin{matrix} & \alpha_1 & \alpha_2 \\ \varphi_1 \\ \varphi_2 \end{matrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad (۶-۷)$$

در این مثال با دریافت پاسخ نامطلوب، خودکار با احتمال 0.8 تغییر حالت می‌دهد و با احتمال 0.2 در حالت فعلی باقی می‌ماند. به این ترتیب ماتریس F (1) غیر قطعی است. در صورتی که F احتمالی باشد، اعضای آن احتمال رفتن به حالت بعدی را با توجه به ورودی و حالت فعلی معین می‌کند. در این صورت ماتریس F می‌تواند به وسیله ماتریس‌های شرطی $F(\beta_m), ..., F(\beta_1)$ که اعضای آن مطابق رابطه زیر تعریف می‌شوند، بیان می‌شود:

$$\begin{cases} i = 1, 2, ..., s \\ j = 1, 2, ..., s \\ \beta = \beta_1, ..., \beta_m \end{cases} f_{ij}^{\beta} = \Pr\{\varphi(n+1) = \varphi_j \mid \varphi(n) = \varphi_i, \beta(n) = \beta\} \qquad (۷-۷)$$

[۱] stochastic

اگر G ماتریس احتمالی باشد، به وسیله یک ماتریس $s \times r$ با اعضای مطابق رابطه زیر قابل بیان می‌باشد:

$$\begin{cases} i = 1,2,...,s \\ j = 1,2,...,r \end{cases} g_{ij} = \Pr\{\alpha(n) = \alpha_j \mid \varphi(n) = \varphi_i\} \qquad (۸-۷)$$

از آنجاکه f_{ij}^{β} و g_{ij} مقادیر احتمالی هستند، در بازه [0, 1] قرار دارند. چون در هر حالت و به ازای هر ورودی، خودکار به طور قطع به حالت بعدی می‌رود و یک عمل به عنوان خروجی انتخاب می‌شود. پس:

$$\sum_{j=1}^{s} f_{ij}^{\beta} = 1 \quad for\ each \quad \beta \in \beta,\ i \in \{1, ..., s\} \qquad (۹-۷)$$

$$\sum_{j=1}^{s} g_{ij}^{\beta} = 1 \quad for\ each \quad i \in \{1, ..., s\}$$

یعنی مجموع عناصر هر سطر باید برابر یک باشد. می‌توان نشان داد که برای هر خودکار با تابع خروجی غیرقطعی، می‌توان خودکار معادل دیگری با خروجی قطعی تعریف نمود.

۷-۳-۳ خودکارهای با ساختار ثابت و متغیر

اگر مقادیر f_{ij}^{β} و g_{ij} در طول عملکرد خودکار ثابت باشند و برحسب زمان و ورودی تغییر نکنند، خودکار مربوط، خودکار با ساختار ثابت[۱] و در صورتی‌که f_{ij}^{β} و g_{ij} در هر لحظه n و یا بر حسب ورودی تغییر کنند، خودکار مربوطه، خودکار با ساختار متغیر[۲] نامیده می‌شود. یک خودکار با ساختار متغیر در واقع یک خودکار غیرقطعی است که مقادیر احتمال‌های مربوطه با توابع تغییر حالت و خروجی در طول زمان تغییر می‌کند.

۷-۳-۴ بردارهای احتمال عمل و حالت

در صورتی‌که احتمال‌های انتقال حالت خودکار معلوم باشند، رفتار خودکار به ازای رشته‌ای از ورودی‌ها قابل پیش بینی می‌باشد. اما در بسیاری موارد دانستن احتمال قرار گرفتن خودکار در هر یک از حالات می‌تواند مفید باشد. به این احتمال‌ها، احتمال‌های کلی-حالت[۳] گفته می‌شود و می‌توانند به نوعی بیان‌گر نحوه

---

[۱] fixed structure
[۲] variable structure
[۳] total state probabilities

عملکرد خودکار باشند. اگر $\pi(n) = [\pi_1(n), ..., \pi_s(n)]^T$ بردار احتمال حالت خودکار باشد، در این صورت:

$$\pi_j(0) = \Pr\{\varphi(0) = \varphi_j\} \qquad (۹-۷)$$

$$\pi_j(n) = \Pr\{\varphi(n) = \varphi_j \mid \beta(0) \ldots \beta(n-1)\}$$

$\pi_j(n)$ احتمال قرار گرفتن خودکار در حالت φ_j در لحظه n می‌باشد. با توجه به تعریف ماتریس تغییر حالت خواهیم داشت:

$$\pi(n) = F^T(\beta(n-1).F^T(\beta(n-2)) \ldots\ldots F^T(\beta(0))\pi(0) \qquad (۱۰-۷)$$

بنابراین بردار احتمال حالت $\pi(n)$ در هر لحظه بر اساس حالت اولیه $\pi(0)$ و ورودی‌های $\beta(i), i = 1, ..., n-1$ معین می‌شود. به همین ترتیب بردار احتمال عمل بیان‌گر احتمال انتخاب عمل α_i در لحظه n می‌باشد به صورت زیر تعریف می‌شود:

$$P(n) = [P_1(n), ..., P_r(n)]^T \qquad (۱۱-۷)$$

$$P_i(n) = \Pr\{\alpha(n) = \alpha_i \mid \beta(0), ..., \beta(n-1)\} \quad i = 1, ..., r$$

با بسط $P_i(n)$ بر حسب خودکار خواهیم داشت:

$$P(n) = G^T \pi(n) \qquad (۱۲-۷)$$

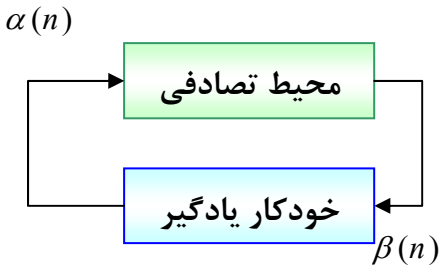
۷-۳-۵ ورودی تصادفی و خودکار یادگیر

در صورتی که یک خودکار تصادفی با ورودی دودویی تصادفی $\beta(n) = 1$ با احتمال c و $\beta(n) = 0$ با احتمال 1-c عمل کند، می‌توان نشان داد که رفتار این خودکار مانند یک خودکار تصادفی با ماتریس تبدیل حالت $\overline{F}$ است که در محیطی با ورودی ثابت عمل می‌کند به طوری که مولفه‌های ماتریس تبدیل حالت آن به صورت رابطه $\overline{f_{ij}} = f_{ij}^{o}(1-c) + f_{ij}^{1}c$ می‌باشد. در صورتی‌که خودکار غیر قطعی با ساختار ثابت باشد، f_{ij}^{β} مقدار ثابتی است، بنابراین $\overline{f_{ij}}$ نیز مقداری ثابت است. بنابراین یک خودکار غیر قطعی با ساختار ثابت که در یک محیط تصادفی فعالیت می‌کند، مانند خودکار با ساختار ثابت غیرقطعی دیگری با ماتریس تغییر حالت $\overline{F}$ می‌باشد که با ورودی ثابت عمل می‌کند. بنابراین این خودکار با خودکار با ساختار متغیر دیگری با ماتریس تغییر حالت $\overline{F}$ می‌تواند مدل شود.

با توجه به تعریف ماتریس تبدیل حالت، اگر احتمال قرار گرفتن خودکار در لحظه n، $\pi(n)$ باشد، بردار احتمال حالت آن در لحظه n + 1 یعنی $\pi(n+1)$ به صورت $\pi(n+1) = F^T \pi(n)$ خواهد بود. بنابراین $\pi(n)$ مانند یک بردار تصادفی عمل می‌کند که به حالت یک لحظه قبل از آن بستگی دارد. بنابراین رشته تغییر حالت‌های یک خودکار با ساختار ثابت در یک محیط ایستا می‌تواند به وسیله یک زنجیره مارکوفی با ماتریس انتقال $\overline{F}$ بیان شود.

۴-۷ رفتار متقابل محیط و خودکار یادگیر

یک خودکار در یک مدار بازخورد[۱] با محیط، تراکنش انجام می‌دهد، به طوری که در هر لحظه خروجی $\beta(n)$ محیط، ورودی خودکار و عمل $\alpha(n)$ انتخاب شده به وسیله خودکار، ورودی محیط می‌باشد. با شروع از حالت اولیه $\varphi(0)$، خودکار عمل $\alpha(0)$ را انتخاب و به محیط اعلام می‌کند و محیط نیز در پاسخ، $\beta(0)$را باز می‌گرداند. خودکار بر اساس پاسخ مثبت به دست آمده به حالت $\varphi(1)$ می‌رود. این چرخه انتخاب عمل، دریافت پاسخ و تغییر حالت ادامه پیدا می‌کند. در خودکارهای با ساختار متغیر، بردار احتمال عمل P(n) با ماتریس تغییر حالت F(n) با تغییر حالت n به‌هنگام می‌شود. این خودکار که به منظور بالا بردن شاخص کارآیی تعریف شده و با محیط ناشناس رفتار کنش-واکنش انجام می‌دهد، خودکار یادگیر نامیده می‌شود. شکل ۷-۵ رفتار متقابل خودکار یادگیر و محیط را نشان می‌دهد [۲].



شکل ۷-۵ : رفتار متقابل خودکار یادگیر و محیط [۲].

در یک محیط با خروجی دودویی (مدلP) خودکار عملی را انتخاب و به محیط اعلام می‌کند و پاسخ مطلوب یا نامطلوب را دریافت می‌نماید. خودکار از مشخصات داخلی محیط (c_iها) آگاهی ندارد. هدف از این فعالیت، یافتن عملی است که احتمال دریافت پاسخ نامطلوب از جانب محیط حداقل باشد. به این عمل، عمل بهینه گفته می‌شود.

---

[۱] feedback

۷-۵ خودکار مهاجرت اشیا (OMA[۱])

خودکار مهاجرت اشیا به صورت پنج‌تایی $< \underline{\alpha}, \underline{\varphi}, \underline{\beta}, \underline{F}, \underline{G} >$ نشان داده می‌شود که $\underline{\alpha} = \{a_1, ..., \alpha_r\}$ مجموعه اقدام‌های[۲] مجاز برای خودکار یادگیر، $\underline{\varphi} = \{\varphi_1, ..., \varphi_s\}$ مجموعه وضعیت‌های[۳] خودکار یادگیر، $\underline{\beta} = \{0, 1\}$ مجموعه ورودی، $F : \underline{\varphi} \times \underline{\beta} \to \underline{\varphi}$ تابع نگاشت وضعیت‌ها و $G : \underline{\varphi} \to \underline{\alpha}$ تابع نگاشت خروجی می‌باشد. این نوع خودکار برای دسته بندی اشیا، انتساب حروف به کلیدها و افراز گراف مورد استفاده قرار می‌گیرد. در این خودکار هر اقدام یک دسته را نشان می‌دهد [۱۵].

در خودکارهای با ساختار ثابت پاسخ محیط به خودکار سبب می‌شود که خودکار از یک وضعیت به وضعیت دیگر منتقل شود، در صورتی که در خودکار مهاجرت اشیا، اشیا به وضعیت‌ها انتساب داده می‌شوند و پاسخ محیط به خودکار سبب گردش اشیا در بین وضعیت‌های خودکار می‌گردد و از طریق این گردش، دسته بندی اشیا انجام می‌گیرد. اگر شی W_i در خروجی jام خودکار مهاجرت اشیا قرار داشته باشد این شی متعلق به دسته شماره j است. برای خروجی α_k مجموعه وضعیت $\varphi_{(k-1)N+1}, ..., \varphi_{kN}$ در نظر گرفته می‌شود که N عمق حافظه[۴] را نشان می‌دهد. به طور کلی می‌توان $\varphi_{(k-1)N+1}$ را داخلی‌ترین وضعیت و φ_{kN} را خارجی‌ترین وضعیت این خروجی دانست. اگر دو شئ W_i و W_j به ترتیب در وضعیت‌های $\varphi_{(k-1)N+1}$ و $\varphi_{(k-1)N+m}$ (برای $m > 1$) قرار داشته باشند، در این صورت قطعیت تعلق شی W_i بیشتر است. بنابراین برای خروجی α_k، وضعیت $\varphi_{(k-1)N+1}$، وضعیت با بیشترین درجه قطعیت و وضعیت ϕ_{kN}، وضعیت با کمترین درجه قطعیت نامیده می‌شود.

در بخش‌های بعدی تعدادی از خودکارهای یادگیر مبتنی بر خودکار مهاجرت اشیا معرفی می‌شوند.

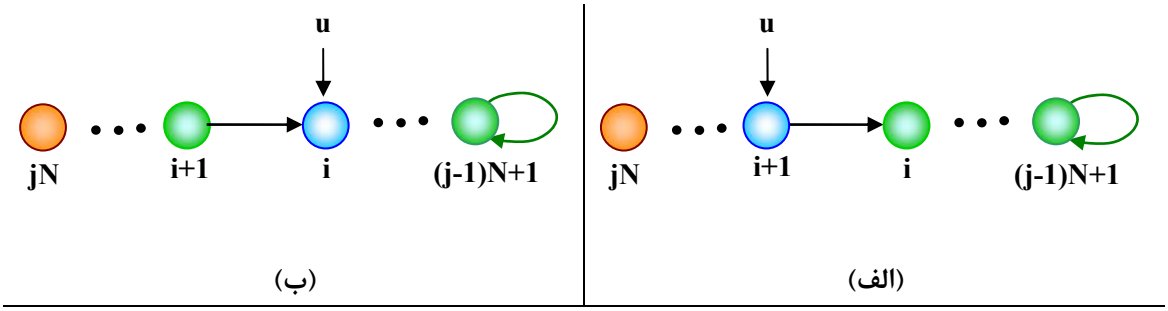
۷-۵-۱ خودکار مهاجرت اشیا مبتنی بر خودکارِ تسلین[۵]

در این نوع خودکار مهاجرت اشیا در هر مرحله از یادگیری، یک اقدام به طور تصادفی برای خودکار انتخاب می‌شود. سپس اقدام انتخاب شده مورد عمل جریمه یا پاداش قرار می‌گیرد. عمل جریمه و پاداش به وسیله یک تابع (که برای هر مساله متفاوت می‌باشد) انجام می‌گیرد. در واقع برای هر مساله باید یک تابع تعریف نمود که مشخص نماید آن اقدامی که انتخاب شده است مورد عمل جریمه قرار می‌گیرد یا مورد عمل پاداش. در زیر نحوه انجام عمل پاداش و عمل جریمه برای خودکار مهاجرت اشیا از نوع تِسلین توضیح داده شده است [۲].

---

۱ Object Migration Automata
۲ action
۳ state
۴ memory depth
۵ Tsetline

(۱) عمل پاداش: اگر اقدام u از خودکار در مجموعه وضعیت‌های $\varphi_{(j-1)N}, \ldots, \varphi_{jN}$ قرار داشته باشد و مورد عمل پاداش قرار بگیرد، در این صورت این اقدام به سمت وضعیت‌های داخلی‌تر خود حرکت می‌نماید. در صورتی که اقدام u در وضعیت $\varphi_{(j-1)N+1}$ قرار داشته باشد، در این صورت با انجام عمل پاداش هیچ تغییری در وضعیت اقدام u انجام نمی‌گیرد. نحوه پاداش دهی به اقدام u در خودکار مهاجرت اشیا از نوع ِتستلین در شکل ۷-۶ نمایش داده شده است.



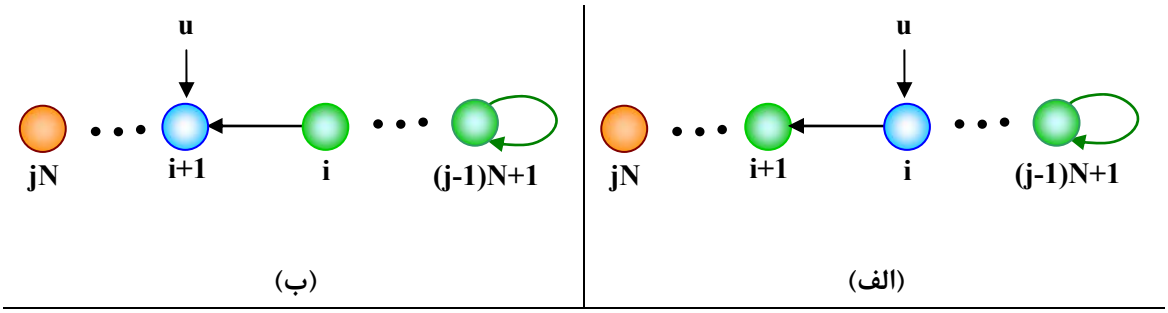
شکل ۷-۶: (الف) وضعیت اقدام u قبل از دریافت پاداش (ب) وضعیت اقدام u بعد از دریافت پاداش [۲].

(۲) عمل جریمه: عمل جریمه شدن اقدام u به دوصورت می‌تواند انجام گیرد:

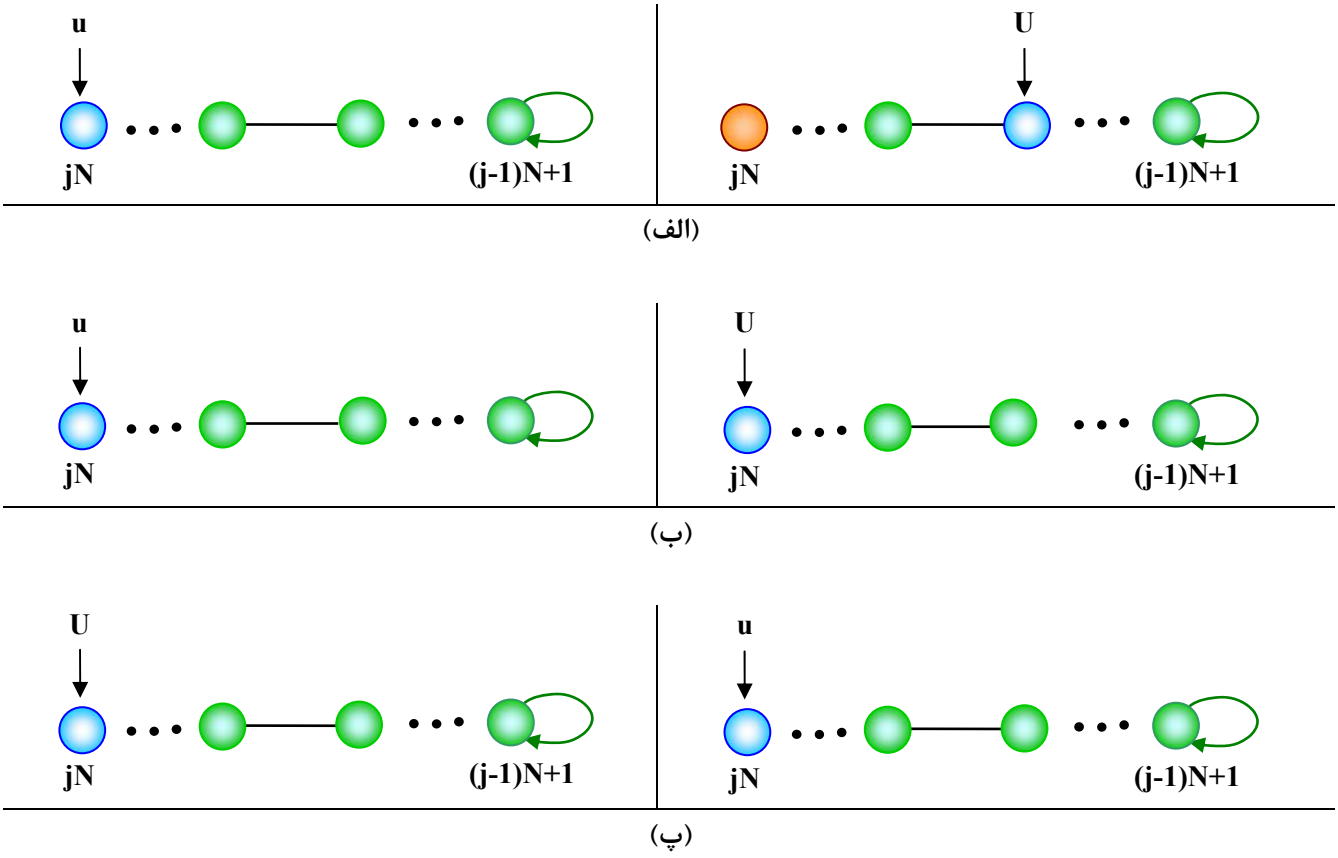
الف) اقدام مورد نظر در مجموعه وضعیت‌های $\varphi_{(j-1)N+1}, \ldots, \varphi_{jN-1}$ قرار دارد: در این صورت جریمه نمودن این اقدام باعث کاهش عمق آن می‌شود. به اصطلاحی اقدام انتخاب شده به سمت وضعیت خارجی‌تر خود تغییر وضعیت داده می‌شود. نحوه جریمه شدن اقدام u در شکل ۷-۷ نمایش داده شده است.

ب) اقدام u در وضعیت φ_{jN} قرار دارد: در این حالت یک اقدام دیگری مثل U از خودکار را پیدا کرده، به طوری که با تعویض کردن مکان‌های اقدام u و اقدام U بهترین نتیجه از خودکار حاصل گردد. در این صورت اگر اقدام U در وضعیت مرزی قرار داشته باشد جای u و U عوض می‌شود و در غیر این صورت ابتدا اقدام U به وضعیت مرزی خود منتقل شده و سپس جابه‌جایی صورت می‌پذیرد. نحوه حرکت چنین اقدامی در شکل ۷-۸ نمایش داده شده است.

الگوریتم شکل ۷-۹ نیز چگونگی پاداش دادن یا جریمه کردن یک اقدام را به طور دقیق نمایش می‌دهد.



شکل ۷-۷: (الف) وضعیت اقدام u قبل از جریمه (ب) وضعیت اقدام u بعد از جریمه [۲].



شکل ۷-۸ : (الف) وضعیت اقدام u قبل از جریمه (ب) انتقال اقدام U به وضعیت مرزی (پ) وضعیت اقدام u بعد از جریمه [۲].

Tsetline Penalize(u) Algorithm	*Tsetline Reward(u) Algorithm*

1.**Begin**
2. if (State(u)% N!=0)
3. inc(State(u));
4. else{
5. bestMakespan=∞;
6. for (U=1;U<=|VG|;U++){
7. create permutation δ' from δ by swapping u and U;
8. if (FT(δ')<bestMakespan){
9. bestMakespan=FT(δ');
10. bestNode=U;
11. }\\end of if
12. }\\end of for
13. }\\end of else
14. State(bestNode)=Action(bestNode)*N;
15. State(u)=Action(u)*N;
16. Swap(State(u),State(bestNode));
17.**End.**
	1. **Begin**
	2. if ((State(u)-1)% N !=0)
	3. dec(State(u));
	4. **End.**

شکل ۷-۹: شبه کد پاداش و جریمه کردن اقدام u در خودکارِ تسلین [۲].

۷-۵-۲ خودکار مهاجرت اشیا مبتنی بر خودکار کرینسکی[۱]

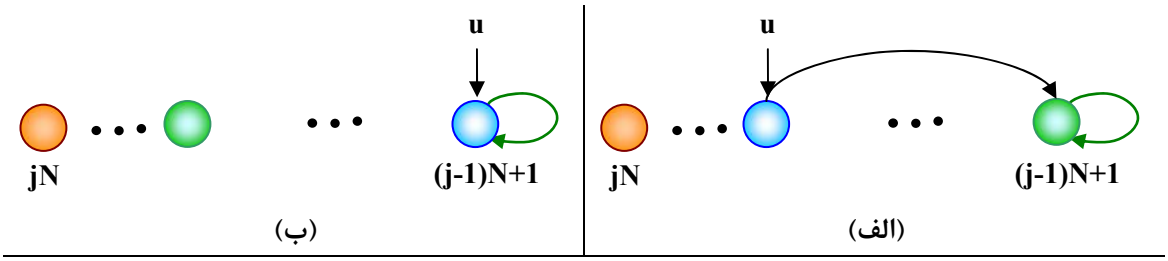
در این نوع خودکار مهاجرت اشیا نیز همانند خودکار تِسلین در هر مرحله از یادگیری، یک اقدام به طور تصادفی برای خودکار انتخاب می‌شود. سپس اقدام انتخاب شده مورد عمل جریمه یا پاداش قرار می‌گیرد. در زیر نحوه انجام عمل پاداش و عمل جریمه برای خودکار مهاجرت اشیا از نوع کرینسکی توضیح داده شده است [۲].

(۱) عمل پاداش: اگر اقدام u از خودکار در مجموعه وضعیت‌های $\varphi_{(j-1)N}$, ..., φ_{jN} قرار داشته باشد و مورد عمل پاداش قرار بگیرد، در این صورت این اقدام به سمت وضعیت‌های داخلی‌تر خود یعنی به وضعیت $\varphi_{(j-1)N+1}$ حرکت می‌نماید. در صورتی که اقدام u در وضعیت $\varphi_{(j-1)N+1}$ قرار داشته باشد، در این صورت با انجام عمل پاداش هیچ تغییری در وضعیت اقدام u انجام نمی‌گیرد. نحوه پاداش دهی به اقدام u در خودکار مهاجرت اشیا از نوع کرینسکی در شکل ۷-۱۰ نمایش داده شده است. در شکل ۷-۱۱ الگوریتم مربوط به عمل پاداش دادن اقدام u برای خودکار مهاجرت اشیا از نوع کرینسکی نمایش داده شده است.

(۲) عمل جریمه: عمل جریمه شدن اقدام u در خودکار مهاجرت اشیا از نوع کرینسکی همانند عمل جریمه در خودکار مهاجرت اشیا از نوع تِسلین است.

---

۱ Krinsky



شکل ۷-۱۰: (الف) وضعیت اقدام u قبل از دریافت پاداش (ب) وضعیت اقدام u بعد از دریافت پاداش [۲].

```
Krinsky Reward(u) Algorithm
1. Begin
2.        State(u)=Action(u)-1)*N+1;
3. End.
```

شکل ۷-۱۱ : شبه کد پاداش دادن به اقدام u در خودکار کرینسکی [۲].

۷-۵-۳ خودکار مهاجرت اشیا مبتنی بر خودکار کرایلو[۱]

در این نوع خودکار مهاجرت اشیا نیز همانند خودکار تِستلین در هر مرحله از یادگیری، یک اقدام به طور تصادفی برای خودکار انتخاب می‌شود. سپس اقدام انتخاب شده مورد عمل جریمه یا پاداش قرار می‌گیرد. در زیر نحوه انجام عمل پاداش و عمل جریمه برای خودکار مهاجرت اشیا از نوع کرایلو توضیح داده شده است [۲].

(۱) **عمل پاداش:** اگر اقدام u در وضعیت $\varphi_{(j-1)N+1}$ قرار داشته باشد با احتمال 0.5 در همین وضعیت باقی می‌ماند و با احتمال 0.5 به وضعیت $\varphi_{(j-1)N+2}$ می‌رود. اگر اقدام u در وضعیت φ_{jN} قرار داشته باشد، با احتمال 0.5 به وضعیت φ_{jN-1} می‌رود و با احتمال 0.5 جریمه می‌شود. در غیر این صورت اقدام u با احتمال 0.5 به وضعیت بعد و با احتمال 0.5 به وضعیت قبل منتقل می‌شود. در الگوریتم شکل ۷-۱۲ شبه کد پاداش دادن به اقدام u نمایش داده شده است.

(۲) عمل جریمه: عمل جریمه شدن اقدام u در خودکار مهاجرت اشیا از نوع کرایلو همانند عمل جریمه در خودکار مهاجرت اشیا از نوع تِستلین است.

---

[۱] Krylov

```
Krylov Reward(u) Algorithm
1. Begin
2.  if (rnd() > 0.5)
3.    if ((State(u)-1)  % N !=0){
4.          dec(State(u));
5.    }\\end of if
6.   else{
7.      if (State(u) % N!=0)
8.         inc(State(u));
9.       else
10.          Penalize(u);
11.   }\\end of else
12. End.
```

شکل ۷-۱۲: شبه کد پاداش دادن به اقدام u در خودکار کرایلو [۲].

۷-۵-۴ خودکار مهاجرت اشیا مبتنی بر خودکار اومن[۱]

در این نوع خودکار مهاجرت اشیا بر خلاف خودکارهای مهاجرت اشیا قبلی در هنگام پاداش دادن، همه اقدام‌های خودکار حرکت می‌کنند. در زیر نحوه انجام عمل پاداش و عمل جریمه برای خودکار مهاجرت اشیا از نوع اومن توضیح داده شده است [۲].

(۱) عمل پاداش: اگر اقدام u از خودکار در مجموعه وضعیت‌های $\varphi_{(j-1)N+1}, \ldots, \varphi_{jN}$ قرار داشته باشد و عمل پاداش انجام گیرد، در این صورت به این جایگشت پاداش داده می‌شود و همه اقدام‌ها به سمت وضعیت‌های داخلی‌تر حرکت می‌کنند. اگر اقدامی در داخلی‌ترین وضعیت اقدام خود باشد، در همان وضعیت باقی می‌ماند. نحوه حرکت اقدام‌ها در الگوریتم شکل ۷-۱۳ نمایش داده شده است.

```
Oommen Reward(u) Algorithm
1. Begin
2.   for (u=1;u<|VG|;u++){
3.          if (State(u)-1)%N!=0)
4.                dec(State(u));
5.    }\\end of for
6. End.
```

الگوریتم ۷-۱۳: شبه کد پاداش دادن به همه اقدام‌ها در خودکار اومن [۲].

---

۱ Oommen

(۲) عمل جریمه: اگر اقدام u از خودکار در مجموعه وضعیت‌های $\varphi_{(j-1)N+1}, ..., \varphi_{jN}$ قرار داشته باشد و مورد عمل جریمه قرار گیرد، در این صورت این جایگشت جریمه شده و همه اقدام‌ها به سمت وضعیت‌های مرزی حرکت می‌کنند. این حرکت آن قدر ادامه پیدا می‌کند تا دست کم یک اقدام از مجموعه اقدام‌ها در وضعیت مرزی قرار گیرد. در این حالت یک اقدام همانند U از خودکار جستجو می‌شود که اگر در جایگشت δ جای u وU عوض شود، بهترین نتیجه در خودکار حاصل شود. اگر اقدام U در وضعیت مرزی قرار داشته باشد، جای u و U عوض می‌شود و در غیر این صورت ابتدا اقدام U به وضعیت مرزی خود منتقل و سپس جابه‌جایی صورت می‌گیرد. نحوه حرکت اقدام‌ها در الگوریتم شکل ۷-۱۴ نمایش داده شده است.

```
Oommen Penalize(u) Algorithm
1.Begin
2. do{
3.     for (u=1;u<|VG| ;u++){
4.          if (State(u)%N!=0)
5.             inc(State(u));
6.       }\\end of for
7.   }while(at least one node appears in the boundary state);
8.  bestMakespan=∞ ;
9.  for (u=1;u<|VG| ;u++){
10.       Create permutation δ′ from δ by swapping u and U
11.       if (FT(δ′)<bestMakespan){
12.            bestMakespan=FT(δ′);
13.            bestNode=U;
14.         }\\end of if
15.     }\\end of for
16.    State(bestNode)=Action(bestNode)*N;
17.    State(u)=Action(u)*N;
18.    Swap(State(u),State(bestNode));
19. End.
```

شکل ۷-۱۴ : شبه کد جریمه کردن اقدام u در خودکار اومن [۲].

۷-۶ ملاک‌های کارآیی

برای قضاوت در مورد یادگیری خودکار، باید ملاک‌های کمی رفتاری تعریف شود. ارایه ملاک‌های عمومی می‌تواند پیچیده باشد. ملاک‌های تعریف شده در این بخش بر اساس محیط با مدلP و با مشخصات ایستا تعریف شده‌اند. هم‌چنین این ملاک‌ها بر اساس بردار احتمال عمل خودکار تعریف شده‌اند. اگر هیچ اطلاعاتی درباره محیط در دست نباشد، ملاکی برای انتخاب عمل‌های α_i، $i = 1, ..., r$ وجود نخواهد د اشت. در این شرایط، عمل‌ها با احتمال‌های مساوی انتخاب می‌شوند. بنابراین مولفه‌های بردار احتمال عمل به صورت زیر خواهد بود:

$$P_i(n) = \frac{1}{r} \quad i = 1, ..., r \qquad (۱۳-۷)$$

چنین خودکاری، خودکار به طور کامل شانسی[۱] نامیده می‌شود. هر خودکار یادگیر، باید کارآیی بهتری نسبت به خودکار به طور کامل شانسی داشته باشد. فرض می‌شود خودکارها در محیطی ایستا با مشخصات داخلی $\{c_1, ..., c_r\}$ فعالیت می‌کنند. اگر دو خودکار در این محیط فعالیت کنند خودکاری که پاسخ صحیح بیشتری از جانب محیط دریافت می‌کند، کارآیی بالاتری دارد. ملاک M (n) به صورت زیر تعریف می‌شود:

$$M(n) = E[\beta \mid p(n)] = \Pr[\beta(n) = 1 \mid p(n)] = \sum_{i=1}^{r} \Pr[\beta(n) = 1 \mid \alpha(n) = \alpha_i].\Pr[\alpha(n) = \alpha_i] \qquad (۱۴-۷)$$

$$\Rightarrow M(n) = \sum_{i=1}^{r} c_i p_i(n)$$

M (n) احتمال دریافت پاسخ نامطلوب در لحظه n می‌باشد. در خودکار به طور کامل شانسی، M (n) مقدار ثابت M_0 به صورت $M_0 = \frac{1}{r}\sum_{i=1}^{r} c_i$ را خواهد داشت. خودکار با کارآیی بهتر، باید حداقل، زمانی که $n \to \infty$، میانگین دریافت پاسخ نامطلوب آن کمتر از M_0 باشد. از آن‌جا که P(n) و $\lim_{n\to\infty} p(n)$ و M(n) و $\lim_{n\to\infty} M(n)$ متغیرهای تصادفی هستند $E[M(n)]$ با M_0 مقایسه می‌شود.

$$E[M(n) = E\{E \mid \beta(n) \mid p(n)]\} = E[\beta(n)] \qquad (۱۵-۷)$$

هر خودکار که کارآیی بهتری نسبت به خودکار به طور کامل شانسی داشته باشد expedient نامیده می‌شود.

تعریف ۷-۱: یک خودکار expedient است اگر $\lim_{n\to\infty} E[M(n)] < M_0$

حداقل مقدار M (n) با توجه به این که $\sum_{i=1}^{r} P_i(n) = 1$ به صورت زیر می‌باشد:

$$\inf M(n) = \inf_{p(n)} \left\{ \sum_{i=1}^{r} c_i p_i(n) \right\} = \min_i \{c_i\} \stackrel{\Delta}{=} c_l \qquad (۱۶-۷)$$

تعریف ۷-۲: یک خودکار یادگیر بهینه است اگر $c_1 = \min_i \{c_i\}$ و $\lim_{n\to\infty} E[M(n)] = c_1$

---

[۱] pure-chance automata

طبق معادله قبل به وسیله خودکار بهینه، درحد، عمل α_1 متناظر با c_1 با احتمال ۱ انتخاب می‌شود. ممکن است رسیدن به حالت بهینه میسر نباشد. در این حالت، ملاک نزدیک به بهینه[۱] تعریف می‌گردد.

تعریف ۷-۳: یک خودکار یادگیر، نزدیک به بهینه است اگر برای هر مقدار مناسب $\varepsilon > 0$ با توجه به پارامترهای خودکار داشته باشد:

$$\lim_{n \to \infty} E[M(n)] < c_1 + \varepsilon \qquad (۱۷-۷)$$

دو فاکتوری که در میزان کارآیی تاثیر می‌گذارند اما به طور واضح در تعریف فوق بیان نشده‌اند، حالت اولیه و مجموعه احتمال‌های خطای محیط هستند. حالت اولیه، حالت شروع عملکرد خودکار می‌باشد و مجموعه احتمال‌های پاسخ منفی خودکار نیز به طور کامل نامعلوم فرض می‌شود. خودکارهایی مطلوب هستند که با شروع از هر حالت اولیه و برای هر محیط ناشناس، رفتار مطلوب از خود نشان دهند. تعریف absolute-expedient چنین ملاکی را تعریف می‌کند.

تعریف ۷-۴: یک خودکار یادگیر absolute-expedient است اگر برای هر n و هر $p_i(n) \in (0, 1)$ و هر مجموعه $\{c_i\}, i = 1, ..., r$:

$$Lim\ E[M(n+1) \mid p(n)] < M(n) \qquad (۱۸-۷)$$

با انجام یک عمل‌گر میانگین در دو طرف معادله فوق خواهیم داشت:

$$E[M(n+1)] < E[M(n)] \qquad (۱۹-۷)$$

یعنی E [M (n)] برای هر محیط تصادفی، پیوسته و نزولی باشد. شرط absolute-expedient شرط قوی‌تری نسبت به expedient است. هم‌چنین می‌توان نشان داد که در محیط‌های تصادفی ایستا، شرط absolute-expedient منجر به $\varepsilon - Optimal$ می‌شود.

---

[۱] $\varepsilon - Optimal$

۷-۷ خودکارهای مرتبط[۱] و بازی‌ها

در مدل یادگیری ارایه شده در مقدمه، یک خودکار منفرد با محیط در تبادل می‌باشد. در واقع یک خودکار منفرد برای حل مساله ساده و ابتدایی مناسب است. از دیدگاه نظری، یک مساله پیچیده را می‌توان با یک خودکار منفرد حل کرد، اما در این صورت ممکن است مجبور به استفاده از خودکاری با تعداد عمل‌های بسیار زیاد شویم که در نتیجه سرعت هم‌گرایی می‌تواند به شدت کاهش یابد. از مفهوم خودکار یادگیر می‌توان هم در جهت ساختن مدل و هم در جهت کنترل کردن به صورت غیر متمرکز[۲] استفاده کرد. در این حالت یک خودکار را می‌توان به عنوان یک بلوک پایه‌ای برای سامانه پیچیده در نظر گرفت. سوالی که مطرح می‌شود این است که آیا می‌توان با استفاده از خودکارهای مرتبط رفتار گروهی را در حل مسایل پیچیده به دست آورد؟ رفتار گروهی خودکارها می‌تواند بسیار پیچیده‌تر از رفتار یک خودکار منفرد باشد.

۷-۷-۱ عدم تمرکز، بازی‌ها و عدم قطعیت

در سامانه‌های پیچیده طبیعی و ساخته دست بشر، عدم تمرکز (توزیع شدگی) امری رایج است. این عدم تمرکز ناشی از عدم توانایی در انتقال کامل اطلاعات بین واحدهای مختلف سامانه می‌باشد که این امر می‌تواند منجر به مشکل شدن هدایت واحدهای تصمیم گیرنده غیر متمرکز در سامانه گردد. نتیجه طبیعی عدم تمرکز، عدم قطعیت، در تصمیم‌گیری است. در چنین سامانه‌ای، اجزا سامانه، اطلاعات کمی از کل سامانه دارند. بنابراین تصمیم‌گیری‌ها تنها براساس اطلاعات محلی باید انجام گیرد. این امر می‌تواند موجب عدم هم‌گونی در بهینه‌سازی محلی و سراسری گردد. تلاش در چنین سامانه‌ای در جهت رفع این ناهم‌گونی صورت می‌گیرد.

اصول اولیه ریاضی در تجزیه و تحلیل مسایل کنترل غیرمتمرکز، از تئوری بازی‌ها[۳] گرفته شده است. در برابر تئوری‌های بهینه‌سازی[۴] که براساس علوم فیزیولوژیکی شکل گرفته است، تئوری بازی برای مواردی که افراد[۵] مختلف اهداف مختلفی را دنبال می‌کنند و هر یک با توجه به ملاک‌های خود عمل می‌کنند، شکل گرفته است. پس این تئوری برای کاربردهایی مانند کاربردهای اجتماعی، اقتصادی و سیاسی که موارد تصمیم‌گیری می‌توانند با یک‌دیگر تضاد داشته باشند، مناسب است.

هر بازی بیش از یک بازیگر و برآمد (نتیجه) - که بستگی به رفتار بازیگران دارد- تشکیل شده است. از آن‌جاکه هر بازیگر، شاخص کارآیی خود را دارد، برآمد بازی به طرق مختلفی به وسیله بازیگران بازی،

---

۱ interconnected automata
۲ decentralized
۳ game theory
۴ optimization theory
۵ agent

می‌تواند سنجیده شود. نحوه ارتباط بازیگران با یک‌دیگر، اطلاعات موجود برای هر بازیگر، ارتباط بین بازی بازیگرها و برآمد بازی به همراه هر نوع توافق ساختاری که بازیگران براساس آن می‌توانند در بازی شرکت کنند قواعد بازی را تشکیل می‌دهند. یک بازی می‌تواند به صورت یک ساختار تصمیم‌گیری تک سطحی یا چند سطحی پیاده‌سازی شود. پیچیدگی بر اساس تعداد بازیگرها (۲ یا n)، شاخص کارآیی (common-payoff و non-zero sum و zero sum) و طبیعت ارتباط بازیگرها (cooperative یا non-cooperative) تقسیم‌بندی می‌شود. اغلب فرض می‌شود عدم قطعیت هر بازیگر ناشی از حضور سایر بازیگرانی است که در بازی هستند و از دید بازیگر ناشناس هستند. اما در مسایل پیچیده‌تر منابع دیگری از عدم قطعیت نظیر پارامترهای سامانه و رخدادهای خارجی می‌تواند وجود داشته باشد. این مسایل می‌تواند باعث مشکل‌تر شدن مساله کنترل بازی و ترکیب تئوری بازی و یادگیری شود.

در بازی خودکارها، با وجود عدم قطعیت زیاد، یک بازی، بارها و بارها تکرار می‌شود. هر بازیگر ممکن است از حضور سایر بازیگران در محیط خبر نداشته باشد و تاثیر بازیگران تنها از طریق اعلام نتیجه بازی از جانب محیط بر یک‌دیگر انجام گیرد. به طور معمول در تئوری بازی‌ها و محیط‌های با مدل P نتیجه 1 به عنوان پاسخ مطلوب در نظر گرفته می‌شود. در هر مرحله هر خودکار از سیاست انتخابی خودش و پاسخ محیط آگاه است و بر اساس این اطلاعات درباره استراتژی بازی بعدی خود تصمیم‌گیری می‌کند.

۷-۷-۲ بازی خودکارها

اگر فرض شود N خودکار $A_N, ..., A_1$ در یک بازی شرکت کرده باشند خودکار A_j به وسیله پنج‌تایی $\{\beta^j, \varphi^j, F^j, \alpha^j, G^j\}$ تعریف می‌شود به طوری که در آن مجموعه ورودی عبارت است از $\beta^j = \{\beta_1{}^j, ..., \beta_{m_j}{}^j\}$، مجموعه حالات عبارت است از $\varphi^j = \{\varphi_1{}^j, ..., \varphi_{s_j}{}^j\}$، تابع تبدیل حالت F^j عبارت است از $\varphi^j(n+1) = F^j[\varphi^j(n), \beta^j(n)]$، مجموعه اعمال خروجی عبارت است از $\alpha^j = \{\alpha_1{}^j, ..., \alpha_{m_j}{}^j\}$ و در نهایت تابع خروجی G^j برابر است با $\alpha^j(n) = G^j[\varphi^j(n)]$.

تعریف ۷-۵ : بازی $\alpha(n)$، مجموعه استراتژی‌هایی است که خودکارها در مرحله n انتخاب می‌کنند و به وسیله یک بردار N تایی نمایش داده می‌شود: $\alpha(n) = [\alpha^1(n), ..., \alpha^N(n)]$

تعریف ۷-۶ : برآمد بازی $\alpha(n)$، یک بردار N تایی $\beta(n) = [\beta^1(n), ..., \beta^N(n)]$ می‌باشد. به طوری که هر $\beta^j(n)$ پاسخ خودکار A_j می‌باشد.

تعریف ۷-۷ : بازی Γ، N خودکار $A_1, ..., A_N$ در یک بازی Γ شرکت کرده‌اند، اگر برآمد $\beta(n)$ بر اساس بازی $\alpha(n)$ باشد. از آنجاکه خودکار A_j، r_j عمل دارد، در مجموع $\prod_{j=1}^{N} r_j$ بازی مختلف وجود دارد و از آن‌جا که هر جزو $\beta^j(n)$ از $\beta(n)$، m_j حالت می‌تواند داشته باشد، متناظر با هر بازی $\prod_{j=1}^{N} m_j$ برآمد وجود دارد. بنابراین برای تعریف کامل یک بازی به $\prod_{j=1}^{N} m_j r_j$ احتمال نیاز داریم. از نظر تئوری محدودیتی روی انواع خودکارهای شرکت کننده در بازی وجود ندارد، برای مثال خودکارهای با ساختار ثابت از یک یا چند نوع، می‌توانند با خودکارهای با ساختار متغیر در یک بازی شرکت داشته باشند. اما از دید سهولت آنالیز بازی، بهتر است خودکارهای شرکت کننده در یک بازی از یک نوع باشند. خودکارها می‌توانند در قالب ساختارهای (پروتکل‌های) مختلف و در سطوح مختلف با یک‌دیگر ارتباط داشته باشند. بر حسب نوع بازی و نیاز مساله می‌توان این ساختار را تعریف نمود. خودکارها می‌توانند همگی در یک سطح قرار گیرند یا در قالب ساختارهای درختی و سلسله مراتبی و هم‌چنین در قالب سامانه‌های مرتبط[۱] با هم‌دیگر مرتبط شوند. یک شیوه بازی، بازی با پاسخ مشترک[۲] می‌باشد که در مسایل شناسایی الگو به طور وسیع مورد استفاده قرار گرفته است. در این شیوه پاسخ محیط برای همه خودکارهای شرکت کننده در بازی یکسان می‌باشد.

۷-۸ کاربردهای خودکارهای یادگیر

مطالعات در زمینه خودکارهای یادگیر هنوز در مراحل ابتدایی خود قرار دارد، با این حال از خودکارهای یادگیر در حل برخی از مسایل استفاده شده است. برخی از کاربردهایی که در آن‌ها از خودکارهای یادگیر استفاده شده است عبارتند از:

۱. مسیریابی در شبکه‌ها: دقت در مسیریابی در هر گروه از شبکه‌های ارتباطی و کاهش میزان ارتباطات بلوک شده و هم‌چنین بهره‌وری متناسب از پهنای باند مسیرهای ارتباطی شبکه (جلوگیری از سرریز اطلاعات در هر گروه) از عوامل کلیدی در کارآیی شبکه‌های ارتباطی محسوب می‌گردد. استفاده از الگوریتم‌های یادگیر در مسیریابی شبکه‌های ارتباطی می‌تواند تاثیر بسیار زیادی در کارآیی هر گروه ارتباطی و در نتیجه کل شبکه داشته باشند. در زمینه مسیریابی در شبکه‌های سوئیچی مداری[۳] و بسته‌ای[۴] و هم‌چنین کنترل جریان داده در هر گروه ارتباطی، مطالعاتی بر اساس خودکارهای یادگیر و بازی خودکارها انجام شده و نشان داده شده که این الگوریتم‌ها با کارآیی خوبی می‌توانند در جهت بالا بردن بازده شبکه، مفید باشند.

---

۱ connectionist systems
۲ play with common pay-off
۳ circuit switching
۴ packet switching

۲. سامانه‌های صف بندی: در یک سامانه پردازش اطلاعات و یا خدمات، در صورتی که متوسط تعداد درخواست‌ها بیشتر از متوسط خدمات باشد، درخواست‌ها در صف‌هایی قرار داده شده و به نوبت بررسی خواهند شد. نحوه مدیریت صف‌ها می‌تواند در کارآیی کل سامانه تاثیر بسیار زیادی داشته باشد. تاثیر خودکارهای یادگیر در برخی سامانه‌های صف مورد بررسی قرار گرفته و نشان داده شده که این الگوریتم‌ها می‌توانند با کارآیی نزدیک به بهینه در سامانه فعالیت کنند.

۳. یکی دیگر از کاربردهایی که در آن از خودکارهای یادگیر استفاده شده، مساله بازشناسی رشته‌های نویزدار[۱] می‌باشد. با استفاده از خودکارهای یادگیر با ساختار ثابت، راه‌حل‌هایی در زمینه حل این مساله ارایه شده است.

۴. در مسایلی مانند پردازش تصویر[۲]، برچسب زنی نمونه‌ها[۳] ، شناسایی و دسته بندی الگوها نیز مطالعاتی با استفاده از خودکارهای یادگیر صورت گرفته و راه‌حل‌هایی ارایه شده است.

۵. در مسایل کنترلی نیز از خودکارهای یادگیر استفاده شده است. یکی از این مسایل، کنترل کوره‌های ذوب کانی‌های معدنی با استفاده از خودکارهای یادگیر سلسله مراتبی می‌باشد. هم‌چنین در زمینه پردازش و کنترل سیگنال‌ها و علائم ارتباطی نیز مطالعاتی با استفاده از خودکارهای یادگیر صورت گرفته است. در زمینه بهینه‌سازی ساختارهای شبکه‌های عصبی با استفاده از خودکارهای یادگیر نیز مطالعاتی صورت گرفته و در زمینه کنترل روباتیک با استفاده از خودکارهای یادگیر نیز مطالعاتی انجام شده است.

---

۱ string taxonomy
۲ image processing
۳ relaxation labeling

۷-۹ رنگ آمیزی گراف با استفاده ازخودکار یادگیر

۷-۹-۱ مقدمه

گراف‌ها یکی از مهم‌ترین ابزارهایی هستند که کاربردهای فراوانی را دارند، یکی از مهم‌ترین کاربردهای گراف‌ها، مساله رنگ آمیزی گراف‌ها می‌باشد. در این مساله برای گراف G به تعداد V گره و E یال وجود دارد. هدف از رنگ آمیزی گراف این است که به هر کدام از گره‌های گراف یک رنگ اختصاص یابد، به طوری که هیچ دو گره مجاوری دارای رنگ یکسان نبوده و از حداقل تعداد رنگ‌ها برای رنگ آمیزی استفاده شود. در میان روش‌های مکاشفه‌ای[۱] غیرقطعی، الگوریتم ژنتیک یکی از مهم‌ترین الگوریتم‌هایی است است که برای رنگ آمیزی گراف مورد استفاده قرار گرفته است. الگوریتم ژنتیک کار خود را با تعدادی جمعیت تصادفی شروع کرده و با تکرار تولید نسل به دنبال کروموزوم شایسته می‌گردد. در الگوریتم ژنتیک هدف پیدا کردن شایسته‌ترین کروموزوم از میان جمعیت‌ها است و به جایگاه ژن‌ها در کروموزوم‌ها هیچ اهمیتی داده نمی‌شود. اگر بتوان جایگاه مناسب ژن‌ها در کروموزوم‌ها را مشخص کرد در این صورت می‌توان در تعداد نسل‌های بسیار کمتری به جواب نزدیک به بهینه رسید. برای پیدا کردن جایگاه مناسب ژن‌ها در کروموزوم‌ها می‌توان از خودکار یادگیر استفاده نمود. خودکار یادگیر با طی فرآیند یادگیری و با اعمال عمل‌گرهای جریمه و پاداش، رفته رفته جایگاه مناسب اقدام‌ها در خودکارها را پیدا می‌کند. در الگوریتم معرفی شده هر کروموزوم در الگوریتم ژنتیک معادل یک خودکار و هر ژن معادل یک اقدام[۲] از خودکار می‌باشد. هر خودکار نمایش دهنده یک رنگ‌آمیزی تصادفی می‌باشد. الگوریتم معرفی شده با تکرار فرآیند یادگیری جایگاه مناسب اقدام‌ها را بهبود می‌بخشد [۷۷].

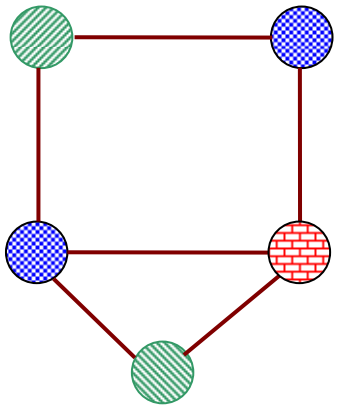
۷-۹-۲ رنگ آمیزی گراف

یک گراف به صورت دوتایی G = (V, E) نشان داده می‌شود که در آن V مجموعه گره‌ها و $E \subseteq V \times V$ مجموعه یال‌ها است. مساله رنگ آمیزی گراف جزو مسایل NP-Complete به شمار می‌رود. در این گونه مسایل با افزایش تعداد گره‌های گراف، زمان مورد نیاز برای پیدا کردن راه حل بهینه، به صورت نمایی رشد پیدا می‌کند. به همین دلیل برای این که راه حل نزدیک به بهینه برای این گونه مسایل پیدا شود، به ناچار از روش‌های جستجوی تصادفی که زمان اجرای آن‌ها به صورت خطی است استفاده می‌شود. راه حل‌های جستجوی تصادفی تضمینی در به دست آوردن بهینه‌ترین جواب ندارند، ولی می‌توانند جواب‌های نزدیک به بهینه را به دست آورند. در این مساله هدف از بهترین جواب عبارت است از حداقل تعداد رنگ‌های مورد نیاز

---

[۱] heuristic
[۲] action

جهت رنگ آمیزی گراف به طوری‌که هیچ دو گرهی هم‌رنگ نباشند. در شکل ۷-۱۵ مثالی از رنگ آمیزی گراف نمایش داده شده است.



شکل ۷-۱۵: مثالی از رنگ امیزی گراف با V = 5 و E = 6 [۷۷].

۷-۹-۳ شرح مساله

برای رنگ‌آمیزی یگ گراف با n گره با استفاده از m رنگ، m^n رنگ‌آمیزی مختلف وجود دارد و در صورتی که از خودکار یادگیر برای حل مساله رنگ‌آمیزی گراف استفاده شود، خودکار یادگیر باید m^n اقدام[۱] داشته باشد که تعداد زیاد اقدام‌ها باعث کاهش سرعت هم‌گرایی می‌شود و به همین منظور از خودکار مهاجرت اشیا به وسیله اومن[۲] و ما[۳] پیشنهاد شده است که یکی از آن‌ها خودکار مهاجرت اشیا مبتنی بر خودکار تِستلین[۴] می‌باشد. الگوریتم ارایه شده از خودکار تستلین برای حل مساله رنگ آمیزی گراف استفاده می‌کند [۲].

در این خودکار $\underline{\alpha} = \{\alpha_1, ..., \alpha_k\}$ مجموعه اقدام‌های مجاز برای خودکاری یادگیر است. این خودکار k اقدام دارد (تعداد اقدام های این خودکار برابر است با تعداد گره‌های گراف). $\underline{\varphi} = \{\varphi_1, \varphi_2, ..., \varphi_{KN}\}$ مجموعه وضعیت‌ها و N عمق حافظه برای خودکار می‌باشد. مجموعه وضعیت‌های این خودکار به K زیر مجموعه $\{\varphi_1, \varphi_2, ..., \varphi_N\}$ و $\{\varphi_{N+1}, \varphi_{N+2}, ..., \varphi_{2N}\}$ و ... و $\{\varphi_{(K-1)N+1}, \varphi_{(K-1)N+2}, ..., \varphi_{kN}\}$ افراز می‌شود و هر کدام از گره‌ها بر اساس این که در کدام وضعیت قرار داشته باشند دسته بندی می‌گردند. در مجموعه وضعیت‌های اقدام k، به وضعیت $\varphi_{(j-1)N+1}$ وضعیت داخلی و به وضعیت φ_{jN}، وضعیت مرزی گفته می‌شود. گرهی که در وضعیت $\varphi_{(j-1)N+1}$ قرار دارد گره با اهمیت بیشتر وگرهی در وضعیت φ_{jN} گره

---

[۱] action
[۲] Oommen
[۳] Ma
[۴] Tsetline

با اهمیت کمتر نامیده می‌شود. در شکل ۷-۱۶ شبه کد مربوط به الگوریتم رنگ آمیزی گراف نمایش داده شده است.

```
Algorithm GCLA (Graph Coloring using Learning Automata)
1. Begin
2. input    G: Graph
3. output   S: a near optimal coloring for G
4.      Generate an initial population, P, of randomly created coloring OMA for G
5.      for (Generation = 1; Generation< Generation. No; Generation++)
6.          for each (automaton, A in P)
7.            Select an action, α_i, randomly
8.             Apply relation 7-20 described in section  7-9-3-2 to reward or penalize α_i
9.          if (rewarded and j < N) upgrade its state from φ_ij to φ_ij+1
10.         else if (penalized and j > 1) downgrade its state from φ_ij+1 to φ_ij
11.          else if (penalized and j = 1) swap the action α_i with α_j  such that  satisfy color α_i
12. End.
```

شکل ۷-۱۶ : شبه کد رنگ آمیزی گراف با استفاده از خودکار یادگیر [۷۷].

در حالت کلی الگوریتم از دو بخش زیر تشکیل شده است:

۱. تولید جمعیت اولیه
۲. عمل‌گر جریمه و پاداش

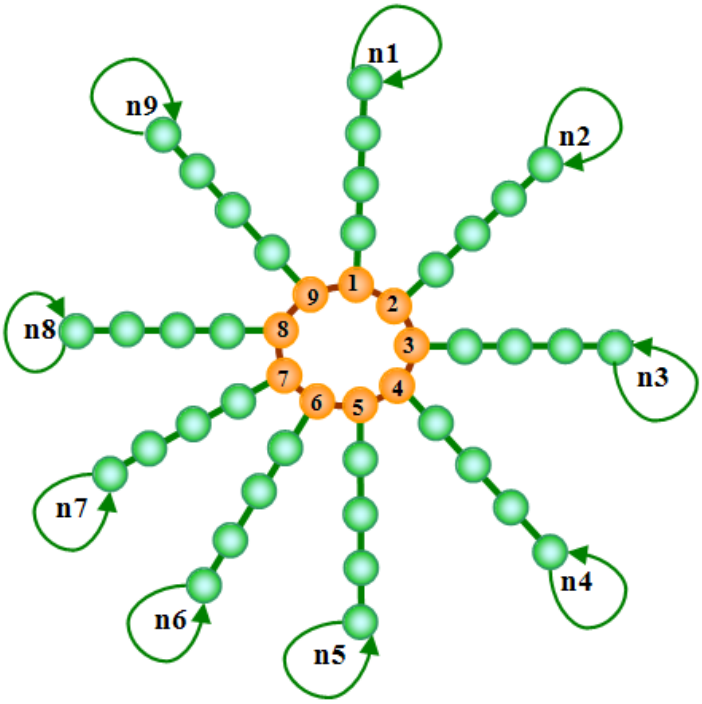
۷-۹-۳-۱ تولید جمعیت اولیه

در ابتدا P خودکار تصادفی ایجاد شده و سپس در هر خودکار گره‌های گراف به اقدام‌های خودکارها تخصیص داده می‌شوند. پس از تخصیص گره‌ها به اقدام‌های خودکار الگوریتم شکل ۷-۱۷ برای تمام خودکارها اجرا می‌شود. با اجرای الگوریتم شکل ۷-۱۷ به هر کدام از اقدام‌ها یک عدد تصادفی اختصاص داده می‌شود. هر عدد نشان دهنده یک رنگ می‌باشد.

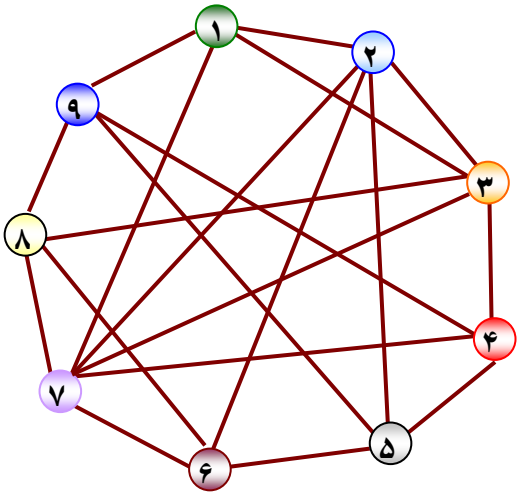
```
Procedure ACA(Assign Colors to Actions)
1. Begin
2. input   N: Nodes
3. output   A: Automata
4.          for (i=1; i<= N ;i++)
5.              assign  i to α_i
6.          for (i=1; i<N/2; i++){
7.            x1=random number
8.            x2= random number}\\end of for
9.            if  (x1!= x2)
10.               Swap (α_x1 value with α_x2)
11.End.
```

شکل ۷-۱۷ : انتساب رنگ‌ها به اقدام‌ها [۷۷].

در شکل ۷-۱۸ یک خودکار تصادفی که نمایش دهنده یک رنگ آمیزی تصادفی می‌باشد، نمایش داده شده است. همان‌طور که مشخص است رنگ آمیزی حاصل از این خودکار در شکل ۷-۱۹ قابل مشاهده است.



شکل ۷-۱۸: یک خودکار تصادفی [۷۷].



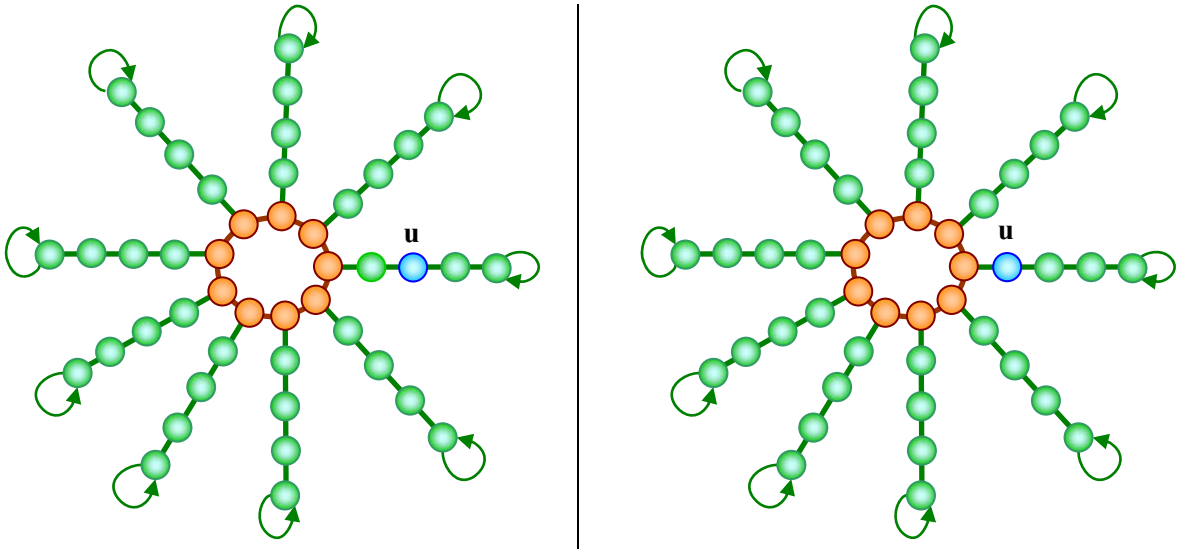
شکل ۷-۱۹: رنگ آمیزی حاصل از خودکار شکل ۷-۱۸ [۷۷].

۷-۹-۳-۲ عمل‌گر جریمه و پاداش

در هر یک از خودکارها یک اقدام به صورت تصادفی انتخاب شده و آن اقدام پاداش یا جریمه می‌شود. در اثر پاداش دادن یا جریمه کردن یک اقدام، وضعیت آن اقدام در مجموعه وضعیت‌های اقدام مورد نظر، تغییر می‌کند. اگر یک اقدام در وضعیت مرزی قرار داشته باشد، جریمه شدن آن باعث تغییر اقدام آن و در نتیجه باعث ایجاد رنگ آمیزی جدیدی می‌شود. عمل‌گر جریمه و پاداش با توجه به نوع خودکاری یادگیر متفاوت خواهد بود.

۷-۹-۳-۲-۱ عمل‌گر پاداش

زمانی که به یک اقدام پاداش داده می‌شود، ممکن است آن اقدام در وضعیت داخلی و یا در وضعیت غیرداخلی قرار داشته باشد. در صورتی که اقدام مورد نظر در وضعیت غیرداخلی قرار داشته باشد در این صورت با پاداش‌دهی به آن اقدام، وضعیت آن به سمت وضعیت داخلی حرکت می‌کند. شکل ۷-۲۰ نحوه پاداش دهی به اقدامی که در وضعیت غیر داخلی قرار دارد را نمایش می‌دهد. در صورتی که اقدام مورد نظر در وضعیت داخلی خود قرار داشته باشد، در این صورت با اعمال عمل پاداش هیچ تغییری در وضعیت آن اقدام صورت نمی‌گیرد و در وضعیت فعلی قرار می‌گیرد.



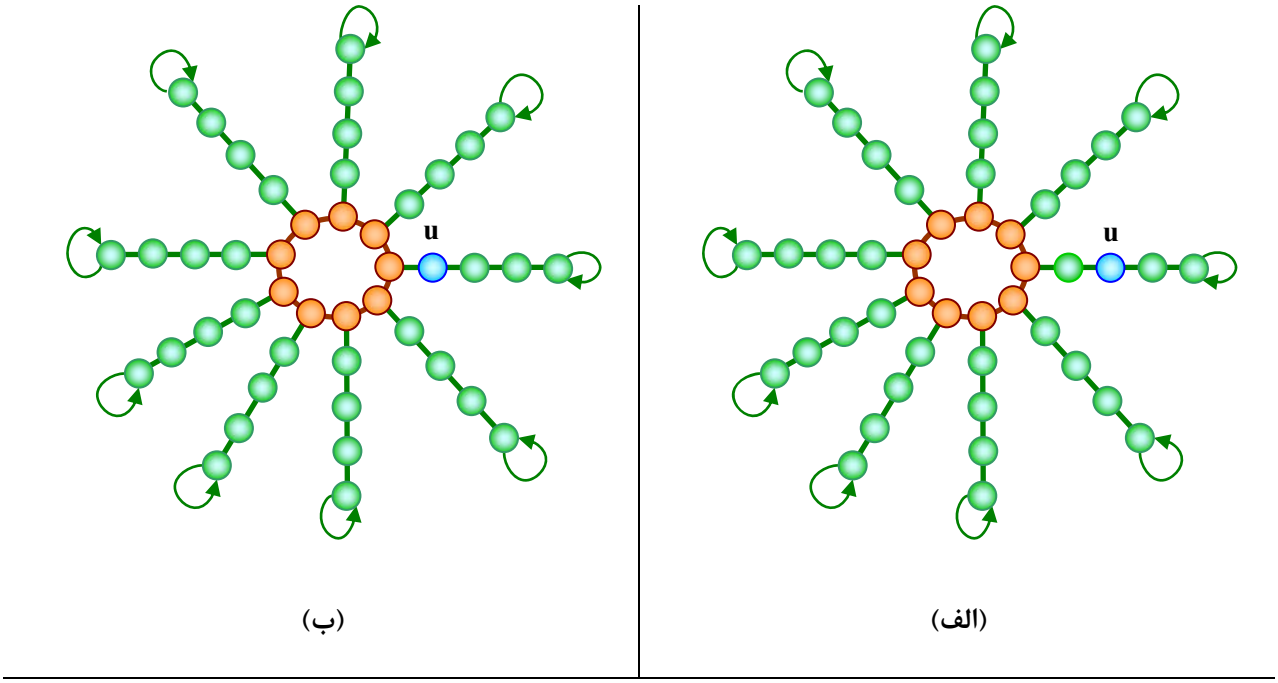
شکل ۷-۲۰ : (الف) وضعیت گره u قبل از دریافت پاداش (ب) وضعیت گره u بعد از دریافت پاداش [۷۷].

۷-۹-۳-۲-۲ عمل‌گر جریمه

موقعی که یک اقدام مورد عمل جریمه قرار می‌گیرد، دو حالت ممکن است اتفاق بیفتد:

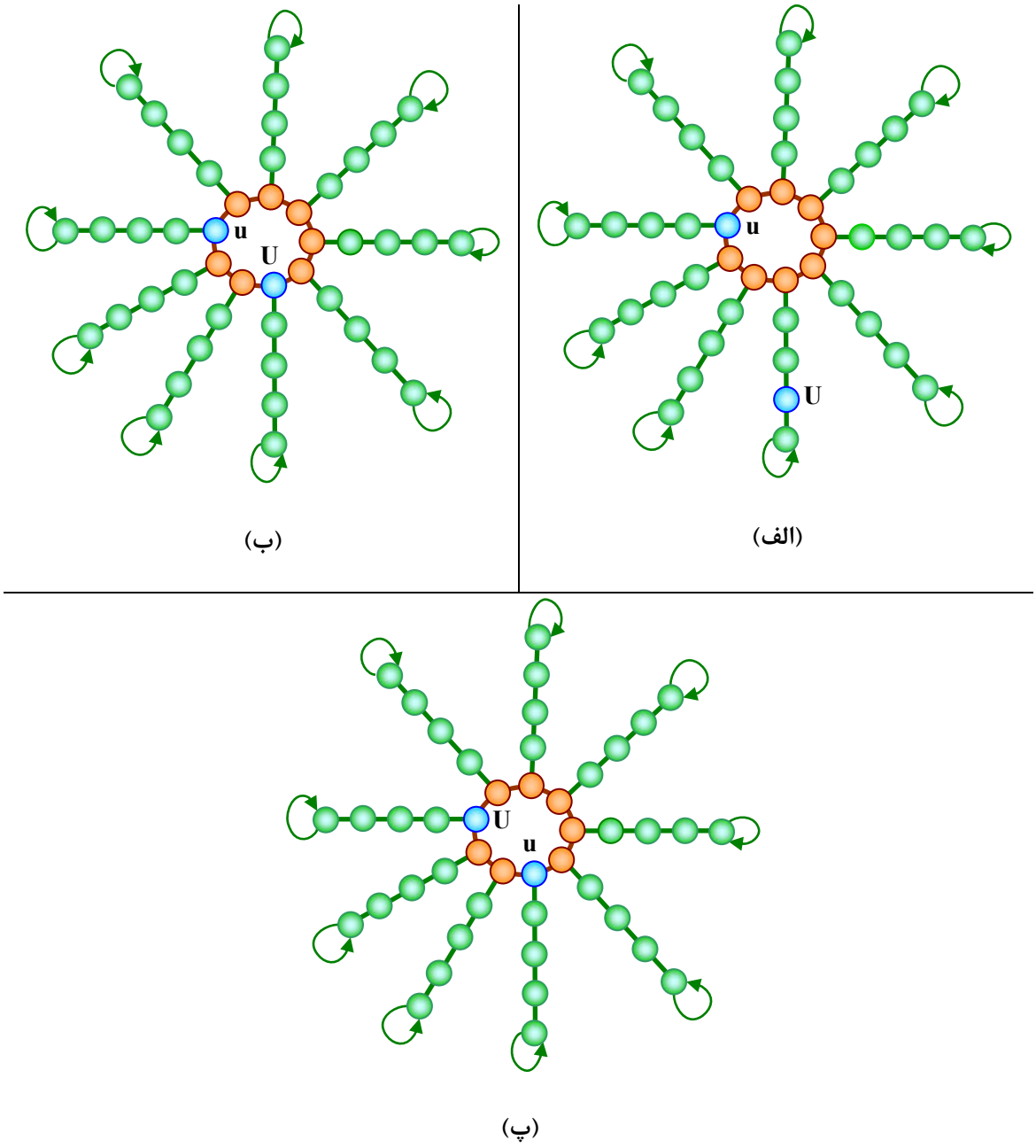
۱. اقدام در وضعیت غیر مرزی قرار داشته باشد
۲. اقدام در وضعیت مرزی قرار داشته باشد.

در صورتی که اقدام مورد نظر در وضعیت غیر مرزی قرار داشته باشد، با اعمال عمل جریمه وضعیت آن اقدام به سمت وضعیت مرزی تغییر داده می‌شود. شکل ۷-۲۱ روال کار را نمایش می‌دهد.



شکل ۷-۲۱ : (الف) وضعیت گره u قبل از جریمه (ب) وضعیت گره u بعد از جریمه [۷۷].

در صورتی که اقدام مورد نظر در وضعیت مرزی قرار داشته باشد، در این صورت مقدار تخصیص داده شده به اقدام مورد نظر (رنگ تخصیص داده شده به اقدام) با عدد تخصیص داده شده به اقدامی جایگزین می‌شود که، پس از جایگزینی آن عدد هیچ دو گره مجاوری همرنگ نباشد. در صورتی که اقدام جایگزین شونده در وضعیت مرزی خود قرار داشته باشد در این صورت مقدار خود را به اقدام جریمه شونده جایگزین می‌کند ولی در صورتی که اقدام جایگزین شونده در وضعیت مرزی خود قرار نداشته باشد، در این صورت در ابتدا به وضعیت مرزی خود انتقال داده شده و پس از آن مقدار خود را به اقدام جریمه شونده جایگزین می‌کند. شکل ۷-۲۲ نحوه جریمه شدن اقدامی را که در وضعیت مرزی خود قرار دارد را نمایش می‌دهد.



شکل ۷-۲۲ : (الف) وضعیت گره u قبل از جریمه (ب) انتقال گره U به وضعیت مرزی (پ) گره u بعد از جریمه [۷۷].

در این بخش مشخص می‌شود که اعمال جریمه و پاداش بر چه اساسی صورت می‌گیرد. یکی از مهم‌ترین کارهایی که در خودکارهای یادگیر مطرح است تعیین رابطه‌ای مناسب برای عمل‌گر جریمه و پاداش است. با در نظر گرفتن گره n_i از گراف با استفاده از رابطه (۷-۲۰) مشخص می‌شود که این گره مورد عمل جریمه قرار گیرد یا مورد عمل پاداش.

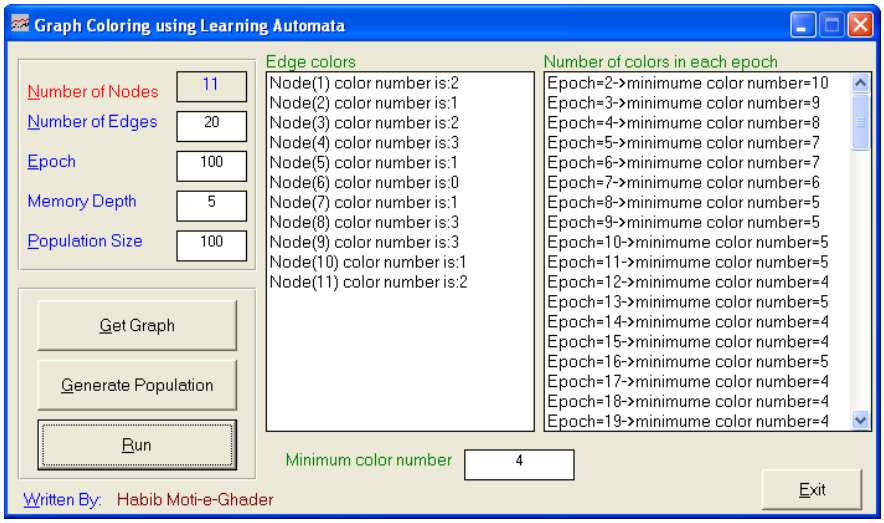
با در نظر گرفتن مفروضات زیر خواهیم داشت:

δ: عبارت است از مجموع تمام رنگ‌های گره‌هایی که با گره n_i در ارتباط هستند.
C: عبارت است از تعداد یال‌های وابسته به گره n_i.
E: عبارت است از تعداد کل یال‌های گراف.

$$n_i = \begin{cases} Reward & if\ (\delta/c \le c/E\ ) \\ Penalize & if\ (\delta/c > c/E) \end{cases} \qquad (۲۰-۷)$$

۷-۹-۴ نتایج حاصل از شبیه‌سازی

برای اجرا و آزمایش نتایج حاصل از الگوریتم معرفی شده یک شبیه‌ساز پیاده‌سازی شده است. این شبیه‌ساز گراف را در قالب یک فایل متنی دریافت کرده و سپس، پس از اجرای الگوریتم تعداد حداقل رنگ‌های مورد نیاز برای رنگ‌آمیزی گراف ورودی را برمی‌گرداند. رابط کاربری شبیه‌ساز پیاده‌سازی شده در شکل ۷-۲۳ قابل مشاهده است. در این بخش نتایج حاصل شده از خودکار یادگیر با نتایج حاصل شده از الگوریتم ژنتیک مورد بررسی و مقایسه قرار داده شده است.



شکل ۷-۲۳: رابط کاربری شبیه‌ساز پیاده‌سازی شده [۷۷].

با در نظر گرفتن گراف‌های شماره ۱ تا ۵ در جدول شکل ۷-۲۴ نتایج حاصل از الگوریتم ژنتیک و خودکار یادگیر قابل مشاهده است. همان‌طور که مشاهده می‌شود نتایجی را که خودکار یادگیر تولید می‌کند، نتایج به تقریب بهتری از نتایج الگوریتم ژنتیک هستند. البته همیشه این طور نیست. مزیتی که

خودکار یادگیر به الگوریتم ژنتیک دارد این است که خودکار یادگیر در تعداد مراحل یادگیر کمتری در مقابل تعداد نسل‌های الگوریتم ژنتیک به جواب بهینه هم‌گرا می‌شود و در به دست آوردن جواب‌های نهایی ممکن است بهتر از ژنتیک عمل کند. ولی به طور معمول نتایج حاصل شده از الگوریتم ژنتیک جواب‌های بهتری از نتایج حاصل شده از خودکار یادگیر می‌باشد.

شماره گراف	تعداد گره	تعداد یال	الگوریتم ژنتیک	خودکار یادگیر
۱	۸۷	۴۰۶	۱۱	۱۰
۲	۷۴	۳۰۱	۱۱	۱۰
۳	۸۰	۲۵۴	۱۰	۹
۴	۱۱	۲۰	۴	۴
۵	۲۳	۷۱	۵	۵

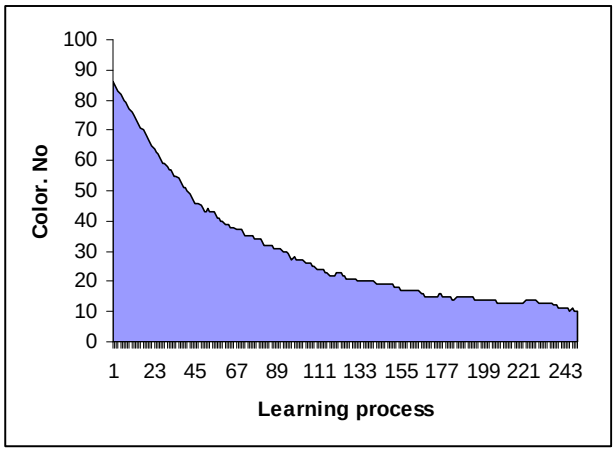
شکل ۷-۲۴ : مقایسه الگوریتم ژنتیک و خودکار یادگیر برای گراف‌های مطرح [۷۷].

پارامترهای خودکار یادگیر که برای شبیه‌سازی گراف‌های شکل ۷-۲۴ بکار رفته شده در جدول شکل ۷-۲۵ نمایش داده شده‌اند.

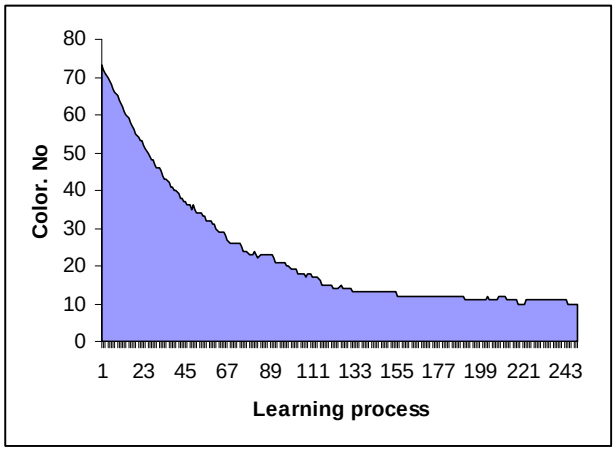
شماره گراف	جمعیت اولیه	تکرار	عمق حافظه
۱	۳۰۰	۲۵۰	۵
۲	۲۵۰	۲۵۰	۵
۳	۲۵۰	۲۰۰	۵
۴	۱۰۰	۵۰	۵
۵	۱۰۰	۱۰۰	۵

شکل ۷-۲۵ : پارامترهای خودکار یادگیر [۷۷].

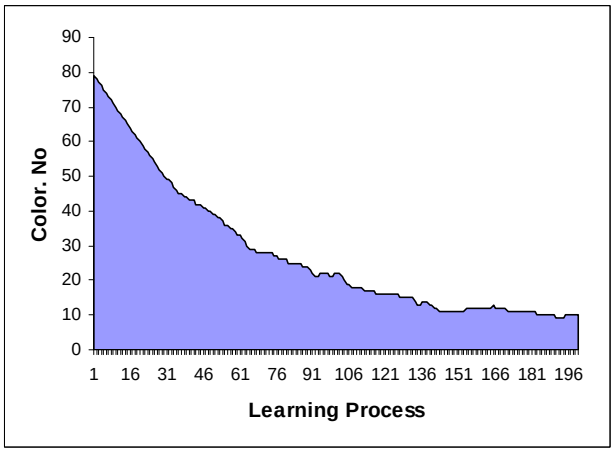
شکل‌های ۷-۲۶ تا ۷-۳۰ سیر یادگیری خودکار یادگیر را برای ۵ گراف نمایش داده شده در شکل ۷-۲۴ را نمایش می‌دهند. همان‌طور که در این شکل‌ها مشاهده می‌شود، خودکار یادگیر در تعداد تکرارهای کمتری به جواب بهینه نزدیک می‌شود یا به عبارتی سرعت هم‌گرایی بالایی دارد.



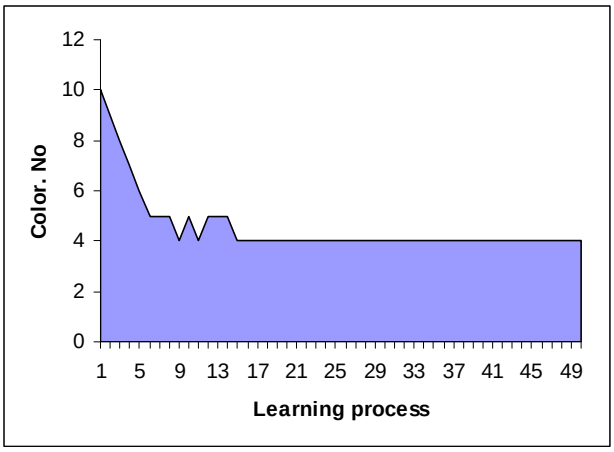
شکل ۷-۲۶ : سیر یادگیری خودکار یادگیر برای گراف شماره ۱ [۷۷].



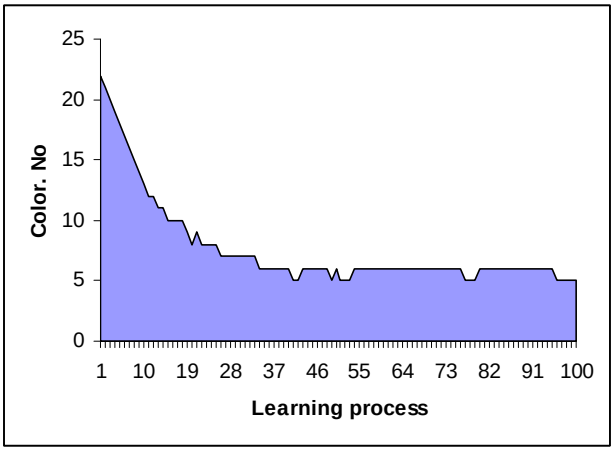
شکل ۷-۲۷ : سیر یادگیری خودکار یادگیر برای گراف شماره ۲ [۷۷].



شکل ۷-۲۸ : سیر یادگیری خودکار یادگیر برای گراف شماره ۳ [۷۷].



شکل ۷-۲۹ : سیر یادگیری خودکار یادگیر برای گراف شماره ۴ [۷۷].



شکل ۷-۳۰ : سیر یادگیری خودکار یادگیر برای گراف شماره ۵ [۷۷].

۷-۱۰ خلاصه فصل

در این فصل در ابتدا خودکارهای یادگیر و انواع آن معرفی شده و سپس خودکارهای مهاجرت اشیا که زیر مجموعه خودکارهای یادگیر می‌باشند شرح داده شدند. پس از آن ملاک‌های کارآیی خودکارهای یادگیر و کاربرد آن در مسایل مختلف توضیح داده شدند. بعد از معرفی خودکارهای یادگیر مساله رنگ آمیزی گراف و نحوه حل این مساله با استفاده از خودکارهای یادگیر شرح داده شده و نتایج حاصل از آن با نتایج حاصل از الگوریتم ژنتیک مورد مقایسه و بررسی قرار گرفتند.

پیوست الف

استفاده از الگوریتم ژنتیک برای حل مساله زمان‌بندی چندپردازنده‌ای*

در این پیوست در ابتدا در بخش الف-۱ زمان‌بندی[۱] چند پردازنده‌ای[۲] معرفی شده و سپس در بخش الف-۲ تعاریف و اصطلاح‌های به کار رفته شده در مساله زمان‌بندی چند پردازنده‌ای مطرح می‌شود. پس از آن در بخش الف-۳ یک الگوریتم ژنتیک جهت زمان‌بندی چند پردازنده‌ای ارایه می‌شود.

الف-۱ زمان‌بندی چند پردازنده‌ای

در مساله زمان‌بندی چند پردازنده‌ای، هدف اجرای یک برنامه موازی روی چندین پردازنده می‌باشد، به طوری که زمان اجرای کل برنامه کمینه گردد. این کار از طریق تخصیص صحیح واحدهای کاری به پردازنده‌ها صورت می‌گیرد. محدودیت‌های اولویتی میان واحدهای کاری نیز باید حفظ گردند. این برنامه به صورت گرافی وزن‌دار و جهت‌دار (DAG[۳]) به نام گراف وظایف[۴] $G = (V, E)$ نشان داده می‌شود. هر گره، عضوی از مجموعه V بوده و یک واحد کاری از برنامه را نشان می‌دهد به طوری که وزن این گره‌ها مشخص کننده زمان اجرای واحد کاری مربوط خواهد بود. این گراف همچنین مجموعه‌ای از یال‌ها، یعنی E را در بردارد که بیان‌گر روابط پیش نیازی بین واحدهای کاری هستند، به این صورت که با داشتن یالی به صورت (t_i, t_j) تا زمانی که t_i اجرایش را به پایان نرساند، t_j نمی‌تواند اجرایش را آغاز کند. این یال‌ها نیز وزن‌دار بوده و وزن هر یال نشان دهنده هزینه ارتباطات و ارسال پیغام میان دو واحد کاری می‌باشد. این هزینه زمانی وجود خواهد داشت که دو واحد کاری مربوطه روی پردازنده‌های مختلف اجرا گردند و در صورتی که پردازنده یکسان باشد، هزینهٔ ارتباطی صفر خواهد بود [۴].

---

* مطالب این پیوست برگرفته شده از مرجع [۲] می‌باشد.

۱ Scheduling

۲ multiprocessor scheduling

۳ Directed Acyclic Graph

۴ task graph

الگوریتم‌های زمان‌بندی در حالت کلی به دو دسته کلی ایستا[۱] و پویا[۲] تقسیم‌بندی می‌شوند:

زمان‌بندی ایستا: در الگوریتم‌های زمان‌بندی ایستا تمام اطلاعات برای زمان‌بندی باید در ابتدا مشخص باشد. این اطلاعات عبارتند از ساختار سامانه، زمان اجرای هر دستورالعمل به صورت مستقل و هزینه ارتباطی بین پردازنده‌ها [۶۹]. روش‌های مختلفی برای تخمین این اطلاعات وجود دارد که می‌توان آن‌ها را در [۷۰] مشاهده کرد. زمان‌بندی ایستا در زمان ترجمه[۳] و قبل از اجرای برنامه موازی انجام می‌گیرد.

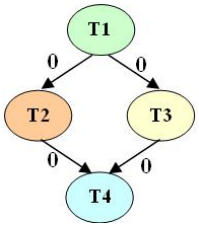
زمان‌بندی پویا: هدف زمان‌بندی پویا، تنها ایجاد یک زمان‌بندی با کیفیت بالا نیست، بلکه سربار زمان‌بندی زمان اجرا را نیز به حداقل می‌رساند. زمان‌بندی پویا در زمان اجرای یک برنامه موازی انجام می‌گیرد [۶۹].

لازم به ذکر است که در این بخش زمان‌بندی ایستا مورد بررسی قرار خواهد گرفت.

برای یک برنامه خاص چندین زمان‌بندی مختلف وجود دارد، ولی باید دقت کرد که هر زمان‌بندی زمان اجرای کل متفاوتی از بقیه دارد. برای مثال در شکل الف-۲-الف و الف-۲-ب دو زمان‌بندی مختلف ممکن برای گراف شکل الف-۱ نمایش داده شده است. همان‌طور که مشاهده می‌شود زمان‌بندی موجود در شکل الف-۲-الف زمان اجرای کل بیشتری نسبت به زمان‌بندی ارایه شده در شکل الف-۲-ب دارد.

زمان‌بندی: به نمودار گانت[۴] حاصل شده از اجرای وظایف بر روی پردازنده‌ها، زمان‌بندی گفته می‌شود.

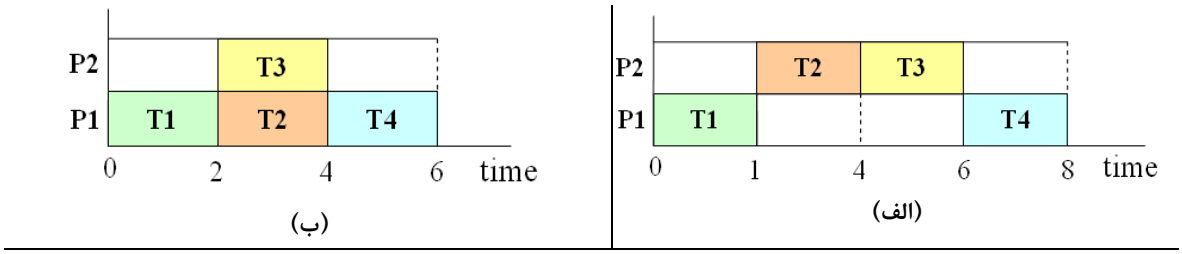
زمان خاتمه: به زمان اتمام آخرین وظیفه از یک گراف وظیفه[۵]، زمان خاتمه زمان‌بندی گفته شده و با FT[۶] نمایش داده می‌شود. گاهی اوقات به زمان خاتمه یک زمان‌بندی makespan نیز گفته می‌شود. برای مثال زمان خاتمه زمان‌بندی موجود در شکل الف-۲-الف برابر 8 و زمان خاتمه زمان‌بندی موجود در شکل الف-۲-ب برابر 6 می‌باشد [۶۹].



شکل الف-۱: مثالی ازگراف وابستگی [۲].

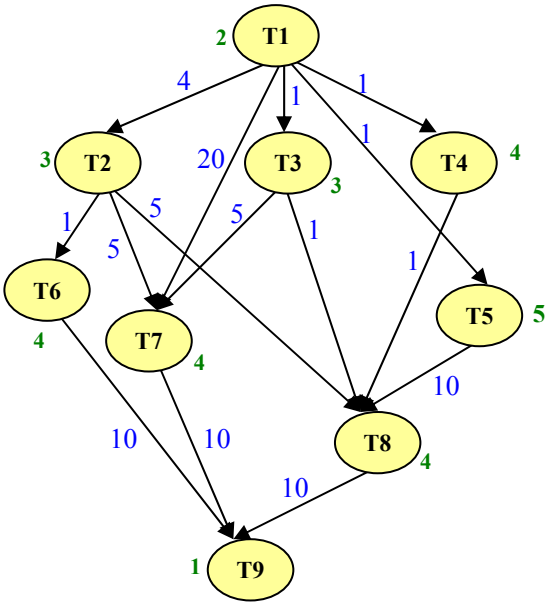
---

۱ static
۲ dynamic
۳ compile
۴ gantt chart
۵ task graph
۶ Finish Time



شکل الف-۲: دو زمان‌بندی مختلف برای مثال شکل الف-۱؛ زمان اجرای هردستورالعمل 2 واحد زمانی می‌باشد [۲].

پیدا کردن زمان‌بندی بهینه یک کار خیلی مشکل بوده و جزو مسایل NP-Hard می‌باشد [۶۷، ۶۹] و نمی‌توان آن را در زمان چندجمله‌ای حل کرد. برای زمان‌بندی یک گراف وظیفه با n گره در روی یک سامانه با p پردازنده، $n! \times p^n$ زمان‌بندی مختلف وجود دارد. این مقدار زمانی درست است که هیچ وابستگی بین وظایف وجود نداشته باشد. در صورتی‌که وابستگی‌های گراف در نظر گرفته شود، تعداد زمان‌بندی‌های ممکن به تقریب برابر $n!$ است. به دلیل این‌که نمی‌توان این مساله را در زمان چندجمله ای حل کرد بنابراین از روش‌های مکاشفه‌ای[۱] برای حل این مساله استفاده شده تا یک زمان‌بندی نزدیک به بهینه را ارایه دهند. در سال‌های اخیر الگوریتم‌های زمان‌بندی مختلفی ارایه شده است. بیشتر الگوریتم‌های ارایه شده، برنامه‌ها را با استفاده از گراف بدون دور جهت‌دار (DAG) نمایش داده‌اند. درشکل الف-۳ مثالی از DAG نمایش داده شده است [۷۱]. وزن‌های تخصیص داده شده به گره‌ها نشان دهنده زمان اجرای[۲] آن وظیفه و وزن‌های تخصیص داده شده به یال‌ها نشان دهنده هزینه ارتباطی بین وظایف می‌باشد.



شکل الف-۳: مثالی از گراف وظایف به همراه هزینه محاسباتی و هزینه ارتباطی [۲].

[۱] heuristic
[۲] computation time

هزینه ارتباطی بین دو وظیفه زمانی مطرح می‌شود که دستوالعمل‌های آن دو وظیفه در پردازنده‌های مختلفی به اجرا در آمده باشند. در صورتی‌که هر دو دستورالعمل بر روی یک پردازنده به اجرا درآید، در این صورت هزینه ارتباطی بین آن دو وظیفه صفر در نظر گرفته می‌شود (در حالت واقعی هزینه ارتباطی صفر نیست بلکه یک زمان بسیارکم می‌باشد. به دلیل خیلی کم بودن این هزینه، آن را صفر در نظر می‌گیرند).

الف-۲ تعاریف و اصطلاح‌های استفاده شده برای زمان‌بندی چند پردازنده‌ای

همان طور که در بخش الف-۱ مطرح شد، برای نمایش یک برنامه از گراف بدون دور جهت‌دار (DAG) استفاده می‌شود. در این بخش برخی از مفاهیم مهم و کلی برای DAG را که در ادامه بیشتر با آن‌ها سروکار خواهیم داشت معرفی خواهد شد. اولویت اجرای وظایف در یک برنامه به وسیله یال‌های موجود در DAG آن برنامه مشخص می‌شود، به طوری که اگر یالی از گره n_i به n_j داشته باشد، در این‌صورت آن یال به صورت e_{ij} نشان داده خواهد شد، به عبارتی دیگر با این کار مشخص می‌شود که اجرای عمل n_j باید پس از اتمام اجرای عمل n_i صورت پذیرد. در واقع محدودیت‌های مورد نیاز برای یک برنامه به وسیله یال‌های موجود در DAG آن برنامه مشخص می‌شود. در زیر تعدادی از اصطلاحات مهم برای DAG معرفی خواهد شد.

الف-۲-۱ طول مسیر

طول مسیر عبارت است از مجموع تمام وزن‌های گره‌ها یا مجموع وزن تمام گره‌ها و یال‌ها از یک گره مبدا تا یک گره مقصد [۶۶]. در صورتی‌که طول مسیر فقط با محاسبه مجموع وزن گره‌ها محاسبه شود، در این صورت به آن طول مسیر محاسباتی[۱] گفته می‌شود.

الف-۲-۲ مسیر بحرانی (CP[۲])

طولانی‌ترین مسیر موجود در بین تمام مسیرهای ممکن در یک DAG را مسیر بحرانی گویند و آن را به صورت CP نمایش می‌دهند. مقدار CP عبارت است از مجموع وزن‌های گره‌ها و یال‌های طول مسیر بحرانی [۶۶]. در یک سامانه موازی کم‌ترین زمان ممکن برای اجرای یک برنامه عبارت است از مقدار مسیر بحرانی. در واقع مسیر بحرانی حداقل زمان اجرای یک برنامه در سامانه موازی را مشخص می‌کند زیرا تمام گره‌های موجود در مسیر CP باید به ترتیب و پشت سر هم اجرا شوند.

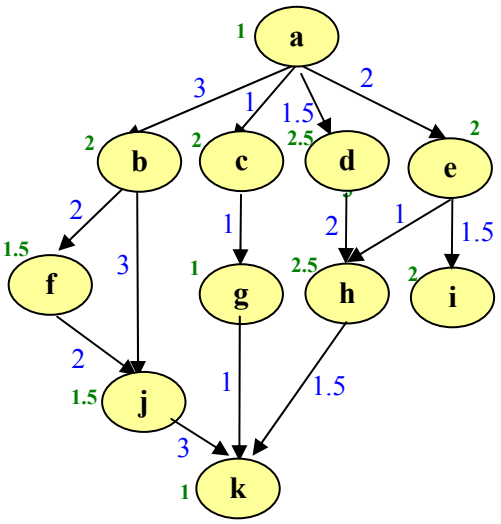
---

[۱] computation path length
[۲] Critical Path

الف-۲-۳ مسیر بحرانی محاسباتی (CP_{comp} [۱])

آن مسیر بحرانی که فقط مبتنی بر وزن گره‌ها می‌باشد، مسیر بحرانی محاسباتی گفته می‌شود [۶۶]. در DAG نمایش داده شده در شکل الف-۴ دو مسیر بحرانی محاسباتی با طول ۷ وجود دارد که عبارتند از: <a, b, f, j, k> و <a, d, h, k>

مسیر بحرانی DAG شکل الف-۴ عبارت است از <a, b, f, j, k> که طول آن ۱۷ می‌باشد. به تمام گره‌های موجود در مسیر بحرانی CPN[۲] گفته می‌شود [۶۶].



شکل الف-۴: مثالی از گراف وظایف به همراه هزینه محاسباتی و هزینه ارتباطی [۲].

الف-۲-۴ سطح یک گره در گراف

در یک گراف سطح هر گره به دو صورت سطح پایین و سطح بالا مشخص می‌شود که در ادامه به تعریف دقیق آن‌ها پرداخته شده است.

الف-۲-۵ گره ورودی[۳] و گره خروجی[۴]

به هر گرهی درگراف که یال ورودی نداشته باشد گره ورودی و به هر گرهی که یال خروجی نداشته باشد گره خروجی گفته می‌شود.

---

[۱] Computation Critical Path
[۲] Critical Path Nodes
[۳] entry node
[۴] exit node

الف-۲-۶ سطح پایین[۱] یک گره

به طولانی‌ترین مسیر موجود در بین تمام مسیرهای موجود میان گره n_i و یکی از گره‌های خارجی، سطح پایین گره n_i گفته می‌شود و به صورت bl (n) نشان داده می‌شود [۶۶].

الف-۲-۷ سطح بالای[۲] یک گره

به طولانی‌ترین مسیر موجود در بین یکی از گره‌های ورودی تا گره n_i را، سطح بالای گره n_i گفته می‌شود و به صورت tl (n) نشان داده می‌شود [۶۶].

نکته۱: در به دست آوردن bl (n) وزن گره n نیز محاسبه می‌شود ولی در محاسبه tl (n) وزن گره n در نظرگرفته نمی‌شود.

پیچیدگی زمانی محاسبه tl و bl برابر $O\ (e + n)$ می‌باشد. که در آن e تعداد یال‌ها و n تعداد گره‌ها می‌باشد. برای درک بیشتر دو تعریف سطح بالا و سطح پایین گره‌ها، مثالی از گراف شکل الف-۴ در زیر ارایه شده است. هدف پیدا کردن سطح پایین و سطح بالای گره b است. ابتدا سطح پایین گره b محاسبه می‌شود از گره b به گره‌های خارجی فقط دو مسیر <b, f, j, k> به طول 13 و <b, j, k> به طول 10.5 وجود دارد. بنا به تعریف سطح پایین یک گره، بزرگ‌ترین مسیر که 13 می‌باشد به عنوان سطح پایین گره b انتخاب می‌شود که آن مسیر عبارت است از <b, f, j, k>. حال سطح بالای گره b محاسبه می‌شود از گره ورودی به گره b تنها یک مسیر <a, b> با طول 4 وجود دارد، که همان نیز برای سطح بالای گره b در نظر گرفته می‌شود. پس در حالت کلی خواهیم داشت:

$$\text{bl (b)} = 13 \quad , \quad \text{tl (b)} = 4$$
$$\text{bl (h)} = 5 \quad , \quad \text{tl (h)} = 7$$

اگر در محاسبه سطح گره n_i فقط از وزن گره‌ها استفاه شود در این صورت، به آن سطح محاسباتی گره n_i گفته می‌شود و به صورت $bl_{comp}\ (n_i)$ و $tl_{comp}\ (n_i)$ نمایش داده می‌شود. برای مثال $bl_{comp}\ (e) = 5.5$ و $tl_{comp}\ (e) = 1$ می‌باشد.

نکته۲: مجموع سطح بالا و سطح پایین هر کدام از گره‌هایی که در مسیر بحرانی قرار دارند برابر است با طول مسیر بحرانی.

---

۱ bottom-level
۲ top-level

$$CP = tl(n_i) + bl(n_i)$$
$$for\ \forall i : n_i \in CP \qquad \text{(الف-۱)}$$

ASAP[۱]**:** عبارت است از نزدیک‌ترین مقدار ممکن برای گره n_i، که معادل همان $tl(n_i)$ می‌باشد [۶۶]. یا به عبارتی:

$$ASAP(n_i) = tl(n_i) \qquad \text{(الف-۲)}$$

ALAP[۲]**:** عبارت است از دیرترین مقدار ممکن برای گره n_i،که معادل $len(cp(n_i)) - tl(n_i)$ می‌باشد [۶۶]. یا به عبارتی:

$$ALAP(n_i) = len(cp(n_i)) - tl(n_i) \qquad \text{(الف-۳)}$$

$len(cp(n_i))$ نشان دهنده طول مسیر بحرانی است. در زیر، در جدول شکل الف-۵ وزن و سطح تمام گره‌های موجود در گراف شکل الف-۴ نمایش داده شده است [۶۶].

nodes	$w()$	$bl()$	$tl()$	$tl() + bl()$	$bl_{comp}()$
a	1	17	0	17	7
b	2	13	4	17	6
c	2	6	2	8	4
d	2.5	9.5	2.5	12	6
e	2	8	3	11	5.5
f	1.5	9	8	17	4
g	1	3	5	8	2
h	2.5	5	7	12	3.5
i	2	2	6	7	2
j	1.5	5.5	11.5	17	2.5
k	1	1	16	17	1

شکل الف-۵: وزن و سطح گره‌های گراف موجود در شکل ب-۴ [۲].

الف-۳ زمان‌بندی چند پردازنده‌ای با استفاده از الگوریتم ژنتیک

مساله زمان‌بندی چند پردازنده‌ای یک مساله جستجو می‌باشد، به طوری‌که فضای جستجو شامل تعداد نمایی از زمان‌بندی‌های مختلف می‌باشد. الگوریتم‌های مبتنی بر جستجو به طور معمول جستجوی خود را با استفاده از یک هیوریستیک محدود می‌کنند. این گونه الگوریتم‌ها تا زمانی عمل جستجو را انجام می‌دهند

[۱] As Soon As possible
[۲] As Late As Possible

که به نتیجه مورد نظر دستیابی پیدا کنند. محاسبات تکاملی[۱] یکی از مقوله‌های مهم الگوریتم‌های مبتنی بر جستجو می‌باشند [۷۳، ۷۶]. الگوریتم شبیه‌ساز سرد کردن فلزات قادر است تا کل فضای مساله را بررسی کند. محاسبات تکاملی از قوانین طبیعت بهره می‌برد. الگوریتم ژنتیک یکی از روش‌های محاسبات تکاملی می‌باشد که برای حل این مساله استفاده شده است. الگوریتم‌های ژنتیک در ابتدا کار خود را با یک جمعیت اولیه[۲] شروع می‌کنند و پس از جستجوی جمعیت اولیه، در صورت نیاز جمعیت جدیدی را تولید کرده و سپس عمل جستجو را در روی جمعیت جدید انجام می‌دهند. تولید نسل تا زمانی انجام می‌شود که به نتیجه مورد نظر حاصل شود.

در این بخش یک روش مناسب برای نمایش رشته‌ها به کار برده شده است. از این رو یک کروموزوم جدید طراحی خواهد شد که نه تنها یک زمان‌بندی کامل را نمایش می‌دهد، بلکه نحوه تخصیص وظایف به پردازنده‌ها را نیز نمایش می‌دهد [۷۱].

الف-۳-۱ چند کروموزومی مبتنی بر اولویت

برای مساله زمان‌بندی چند پردازنده‌ای روش‌های مختلفی با استفاده از الگوریتم‌های ژنتیک ارایه شده است. الگوریتم‌های ژنتیک ارایه شده جهت حل مساله زمان‌بندی چند پردازنده‌ای هر کدام دارای نکات ضعفی در روش‌های کدگذاری دارند. نحوه کدگذاری یک راه حل از مساله به صورت یک کروموزوم، یکی از ویژگی‌های مهم الگوریتم‌های ژنتیک می‌باشد. در ۱۰ سال اخیر، روش‌های کدگذاری مختلفی برای این مساله مطرح شده است. روش کدگذاری دودویی که آخرین بار مطرح شده است، برای مساله زمان‌بندی چند پردازنده‌ای کارآیی بهتری ندارد. مشکلاتی که این روش دارد عبارتند از:

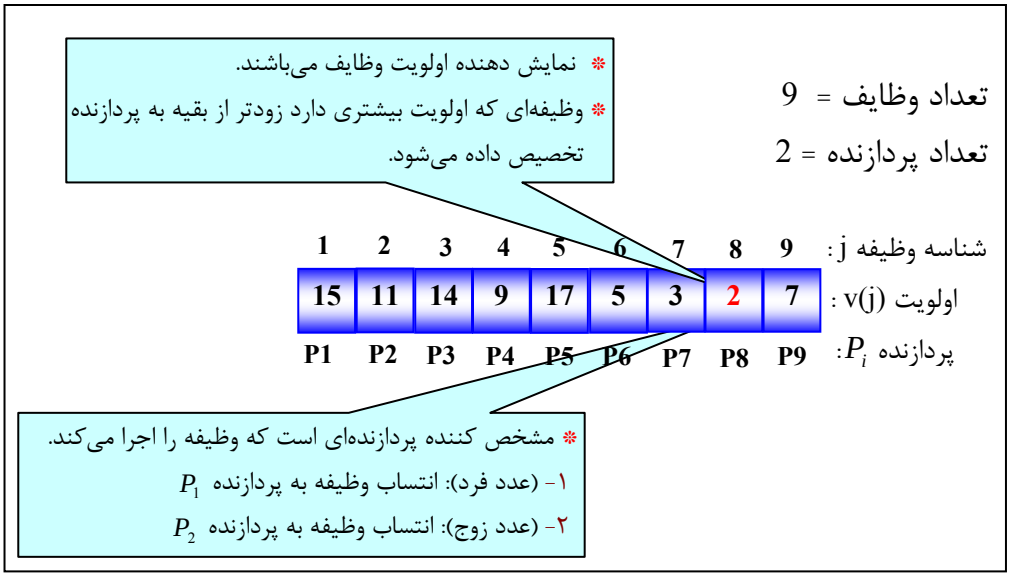
(۱) در مورد زمان‌بندی چند پردازنده‌ای، تعداد زیادی از کروموزوم‌ها برای یک زمان‌بند می‌تواند وجود داشته باشد.

(۲) در مورد زیاد بودن تعداد وظایف، عمل‌گرهای ترکیب و جهش با رعایت اولویت وابستگی‌های وظایف، کار سختی می‌باشد.

برای غلبه بر این مشکل، روش گسترده‌ای به نام PMC[۳] جهت نمایش رشته‌ها به کار برده شده است، که هر کدام از رشته‌ها، اولویت گره‌ها را به همراه پردازنده‌ای که به آن باید تخصیص یابد را نمایش می‌دهد. برای مثال شکل الف-۶ یک PMC ساده برای گراف نمایش داده شده در شکل الف-۳ را با دو پردازنده نمایش می‌دهد.

---

[۱] evolution computation
[۲] population
[۳] Priority bases Multi Chromosome



شکل الف-۶: مثالی از PMC [۲].

در شکل الف-۶ عدد ۲ نمایش دهنده دو حالت مختلف زیر می‌باشد:

۱. اولویت وظیفه را مشخص می‌کند.

۲. نشان دهنده این است که وظیفه جاری به کدام پردازنده تخصیص داده خواهد شد.

در این مثال ۲ پردازنده در نظر گرفته شده است، بنابراین پردازنده‌ها با شماره‌های P1 و P2 مشخص می‌شوند. در صورتی‌که عدد مورد نظر ۱ باشد، گره مورد نظر به پردازنده شماره ۱ تخصیص می‌یابد و در صورتی‌که عدد مورد نظر ۲ باشد، گره مورد نظر به پردازنده شماره ۲ تخصیص می‌یابد. در حالت کلی گره‌هایی که مقدار اولویت آن ها فرد می‌باشد به پردازنده شماره ۱ تخصیص داده می‌شوند و گره‌هایی که مقدار اولویت آن‌ها زوج باشد به پردازنده شماره ۲ تخصیص داده می‌شوند.

الف-۳-۲ کد گذاری و کدگشایی مبتنی بر اولویت

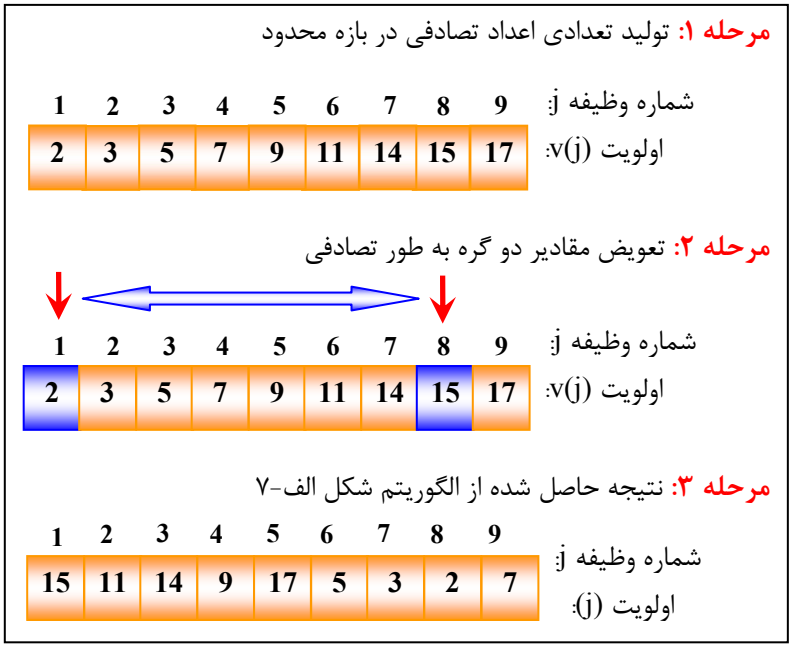
یکی از مراحل مهم این است که چگونه عمل کدگذاری انجام گیرد تا وابستگی بین گره‌ها رعایت شود. روش کدگذاری پیشنهاد شده، مبتنی بر اولویت وظایف می‌باشد. الگوریتم شکل الف-۷ نحوه تولید PMC را نمایش می‌دهد.

```
Priority-Based Encoding Algorithm
1. Begin
2. input: number of processors m, number of tasks n
2. output: chromosome v(j )

3.        for (j = 1; j<=n; j++)
4.              v(j ) ← m * j -random[0,m − 1];
5.        for (i = 1; i<=⌊n/2⌋; i++){
6.              j ← random[1, n];
7.              k ← random[1, n];
8.              if (j ≠ k)
9                  swap (v(j ), v(k));
10.             }\\end of for
11.      output:  chromosome v(j );
12. End.
```

شکل الف-۷: الگوریتم PMC [۲].

در ابتدا تعدادی اعداد تصادفی که وابسته به تعداد پردازنده‌ها است تولید می‌شود. در مثال به کار برده شده برای کدگذاری از ۲ پردازنده استفاده شده است. در مرحله بعدی به تعداد $n/2$ بار به طور تصادفی دو گره انتخاب شده و مقدار اولویت آن‌ها با یک‌دیگر عوض می‌شود. پس از انجام مراحل فوق کروموزوم نهایی حاصل می‌شود. در شکل الف-۸ کروموزوم حاصل شده ازکروموزوم شکل الف-۶ نمایش داده شده است.



شکل الف-۸: کروموزوم حاصل شده از الگوریتم PMC [۲].

حال با استفاده از الگوریتم شکل الف-۹ ترتیب وظایف مشخص می‌شود. فرض کنید که انتظار می‌رود با استفاده از الگوریتم PMC، n وظیفه به m پردازنده انتساب داده شود. با در نظر گرفتن DAG شکل الف-۳ در ابتدا سعی می‌شود تا اولین زمان‌بندی پیدا شود. T_1 تنها وظیفه‌ای است که می‌تواند انتخاب شود. T_2 و T_3 و T_4 و T_5 می‌توانند به عنوان گره بعدی انتخاب شوند، به طوری‌که اولویت آن‌ها به ترتیب عبارتند از: ۱۱، ۱۴، ۹ و ۱۷. بنابراین گره T_5 اولویت بیشتری از بقیه دارد. پس به لیست ترتیب وظایف (TS[۱]) اضافه می‌شود. سومین گره امکان‌پذیر یکی از گره‌های T_2 ، T_3 و T_4 می‌باشد، که به ترتیب دارای اولویت‌های ۱۱ و ۱۴ و ۹ می‌باشند. چون اولویت T_3 بیشتر از بقیه است، انتخاب شده و به لیست اضافه می‌شود. در نهایت مراحل تا زمانی که تمام گره‌ها به لیست ترتیب اضافه می‌شوند انجام داده می‌شود. پس از انجام تمام مراحل لیست ترتیب وظایف به صورت: $TS = (T_1, T_5, T_3, T_2, T_4, T_6, T_7, T_8, T_9)$ حاصل می‌شود.

[۱] Task Sequence

Priority-Based Decoding Algorithm **(one task sequence growth)**
1. **Begin**
2. input: number of tasks *n*, chromosome *v(j)*, the set of nodes $\overline{S}$
3. output: task sequence *TS*
3. $\overline{S} \leftarrow \varnothing$, $TS \leftarrow \varnothing$;
4. $n \leftarrow 0, j \leftarrow 0$;
5. while $(j \leq n)$ {
6. $\overline{S} \leftarrow \overline{S} \cup suc_j$;
7. $j* \leftarrow \arg\max\{v(j)|j \in \overline{S}\}$;
8. $\overline{S} \leftarrow \overline{S} \setminus j*$;
9. $TS \leftarrow TS \cup j*$;
10. $j \leftarrow j*$;
11. }\\end of while
12. output: task sequence *TS*;
13.**End.**

شکل الف-۹: کد گشایی مبتنی بر اولویت [۲].

در مرحله بعدی وظایف از لیست ترتیب وظایف که با استفاده از الگوریتم شکل الف-۹ ایجاد شده، به پردازنده‌های متناظر تخصیص داده می‌شوند. در مثال نمایش داده شده در شکل الف-۶ تنها دو پردازنده بکار رفته است. بنابراین می‌توان تنها دو حالت فرد و زوج را برای مقادیر اولویت‌ها در نظر گرفت. اگر مقدار اولویت یک گره فرد باشد، در این صورت آن گره به پردازنده شماره 1 تخصیص داده می‌شود و اگر مقدار اولویت یک گره زوج باشد، در این صورت آن گره به پردازنده شماره 2 تخصیص داده می‌شود. لذا با در نظر گرفتن مقادیر اولویت گره‌ها در شکل الف-۶ داریم:

$$S = \{p_1 : (T_1, T_5, T_2, T_4, T_6, T_7, T_9),\ p_2 : (T_3, T_8)\}$$

نکته۳: در این مرحله از انتساب باید وابستگی بین گره‌ها در نظرگرفته و هزینه ارتباطی میان وظایف که در پردازنده‌های مختلف اجرا می‌شوند نیز در نظر گرفته شود.

وظیفه الگوریتم شکل الف-۱۰ انتساب وظایف به پردازنده‌ها با استفاده از لیست ترتیب وظایف است که از طریق الگوریتم شکل الف-۹ به دست آمده است.

Priority-Based Decoding Algorithm **(assigning tasks to processors)**

1. **Begin**
2. input: processing time p_k, task sequence *TS*, chromosome *v(j)*, the communication delay τ_{jk}
3. output: makespan *f*, schedule *S*
4. $Pi \leftarrow 0,\ i = 1, 2, \ldots, m;$
5. $t_k \leftarrow 0,\ k = 1, 2, \ldots, n;$
6. $s \leftarrow 0;$
7. for ($j = 1$; j<= *n; j++){*
8. $s \leftarrow TS_i$;
9. $Pi \leftarrow v(s)\%m;$
10. if ($p_i = 0$) $p_i \leftarrow m;$
11. if (assigned task $T_j < T_k$)
12. if ($p_i \neq p_l$ *)*
13. $e_k \leftarrow \max\{t_j \mid j \in \text{pre}(k)\} + \tau_{jk}$;
14. $t_k \leftarrow e_k + p_k$;
15. $S \leftarrow S \cup S_k \{ p_i ; e_k + p_k \};$
16. else $e_k \leftarrow \max\{t_j \mid j \in pre(k)\};$
17. $t_k \leftarrow e_k + p_k$;
18. $S \leftarrow S \cup S_k \{ p_i ; e_k + p_k \};$
19. else $e_k \leftarrow \max\{t_j \mid j \in pre(k)\};$
20. $t_k \leftarrow e_k + p_k$;
21. $S \leftarrow S \cup S_k \{ p_i ; e_k + p_k \};$
22. }\\end of for
23. **o**utput**:** schedule *S*
24. makespan $f \leftarrow \max\{t_k, k = 1, 2, \ldots, n\};$
25. **End.**

شکل الف-۱۰: انتساب وظایف به پردازنده ها [۲].

جدول شکل الف-۱۱-الف و الف-۱۱-ب مراحل انتساب وظایف به پردازنده‌ها را قدم به قدم نمایش می‌دهد.

(الف)

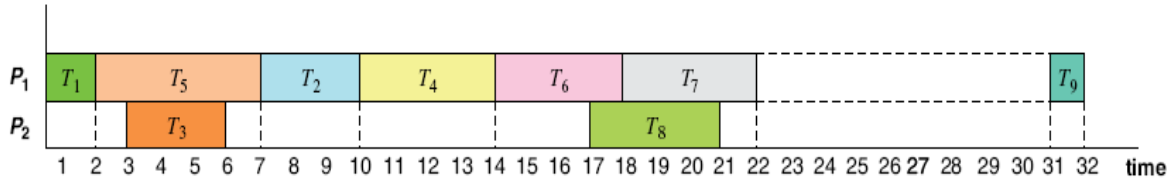
J	$\overline{S}$	$v\ (j)$	J^*	TS
0	{1}	v (1) = 15	1	S = {1}
2	{2, 3, 4, 5}	v (2) = 11, v (3) = 14, (4) = 9, v (5) = 17	5	S = {1, 5}
1	{2, 3, 4}	v (2) = 11, v (3) = 14, (4) = 9	3	S = {1, 5, 3}
3	{2, 4}	v (2) = 11, (4) = 9	2	S = {1, 5, 3, 2}
5	{4, 6, 7}	v (4) = 9, v (6) = 5, v (7) = 3	4	S = {1, 5, 3, 2, 4}
4	{6, 7, 8}	v (6) = 5, v (7) = 3, v (8) = 2	6	S = {1, 5, 3, 2, 4, 6}
7	{7, 8}	v (7) = 3, v (8) = 2	7	S = {1, 5, 3, 2, 4, 6, 7}
9	{8}	v (8) = 2	8	S = {1, 5, 3, 2, 4, 6, 7, 8}
6	{9}	v (9) = 7	9	S = {1, 5, 3, 2, 4, 6, 7, 8, 9}

(ب)

J^*	S	P_i	$t_j = e_j + p_j$
1	P1 = {1}, p2 = {}	1	$t_1 = 0 + 2 = 2$
5	P1 = {1, 5}, p2 = {}	1	$t_5 = 2 + 5 = 7$
3	P1 = {1, 5}, p2 = {3}	2	$t_3 = 3 + 3 = 6$
2	P1 = {1, 5, 2}, p2 = {3}	1	$t_2 = 7 + 3 = 10$
4	P1 = {1, 5, 2, 4}, p2 = {3}	1	$t_4 = 10 + 4 = 14$
6	P1 = {1, 5, 2, 4, 6}, p2 = {3}	1	$t_6 = 14 + 4 = 18$
7	P1 = {1, 5, 2, 4, 6, 7}, p2 = {3}	1	$t_7 = 18 + 4 = 22$
8	P1 = {1, 5, 2, 4, 6, 7}, p2 = {3,8}	2	$t_8 = 17 + 4 = 21$
9	P1 = {1, 5, 2, 4, 6, 7, 9}, p2 = {3,8}	1	$t_9 = 31 + 1 = 32$

شکل الف-۱۱: مراحل انتساب وظایف به پردازنده ها با استفاده از DAG شکل الف-۳ [۲].

در نهایت پس از انتساب وظایف به پردازنده‌ها، زمان‌بندی نمایش داده شده در شکل الف-۱۲حاصل خواهد شد.



شکل الف-۱۲: نمودار گانت حاصل از DAG شکل الف-۳ [۲].

الف-۳-۳ تابع ارزیابی

برای مساله زمان‌بندی چند پردازنده‌ای می‌توان چندین فاکتور را از قبیل توان عملیاتی، زمان خاتمه، کارآیی پردازنده و غیره را برای تابع ارزیابی دخالت داد. در این روش محاسبه تابع ارزیابی خیلی ساده می‌باشد. در ابتدا زمان خاتمه هر کدام از رشته‌ها (زمان‌بندی‌ها) محاسبه می‌شود. طول هر رشته پردازش‌گر برای پیدا کردن زمان اجرای کل زمان‌بندی اندازه گیری می‌شود. تابع ارزیابی که برای این الگوریتم بکار رفته است مبتنی بر makespan (f) زمان‌بندی می‌باشد. تابع ارزیابی به صورت زیر محاسبه می‌شود:

$$eval\,(v_k) = 1 / f^{\,k}\,, k = 1, ..., popsize$$

که $f^{\,k}$ بیان‌گر kامین کروموزوم می‌باشد.

الف-۳-۴ عمل‌گرهای ژنتیک

در این بخش عمل‌گرهای ژنتیک که برای حل مساله زمان‌بندی چندپردازنده‌ای استفاده شده است، معرفی خواهد شد.

الف-۳-۴-۱ عمل‌گر ترکیب

در این بخش یک روش جدید برای ترکیب کروموزم‌ها ارایه می‌شود. روش ترکیبی که در این بخش استفاده شده است، یک روش ترکیب دو نقطه‌ای است. در الگوریتم شکل الف-۱۳ زیر روال مربوط به عمل ترکیب (WMX[۱]) آمده است. در ابتدا به طور تصادفی دو نقطه را به عنوان زیر رشته انتخاب کرده و سپس مقدار و ترتیب آن‌ها بررسی می‌شوند. برای نمونه زیر رشته انتخاب شده از v_1، ترتیب وزنی ۱-۲-۳-۴ دارد. این ترتیب وزنی، برای تغییر زیر رشته انتخاب شده با v_2 بکار می‌رود. بنابراین زیر رشته ۶-۱۳-۱۵-۱۱ به زیر رشته ۱۵-۱۳-۱۱-۶ تغییر می‌یابد و v_2 با ترتیب وزنی زیر رشته v_1، تغییر می‌یابد. الگوریتم WMX، الگوریتمی نیست که فقط مقادیر دو نقطه انتخاب شده از دو کروموزوم را تغییر دهد، بلکه مقادیر رشته‌ها را به ترتیب اولویت وزن‌ها تغییر می‌دهد. در شکل الف-۱۴ مثالی برای ترکیب کروموزوم‌ها با استفاده از الگوریتم WMX نمایش داده شده است.

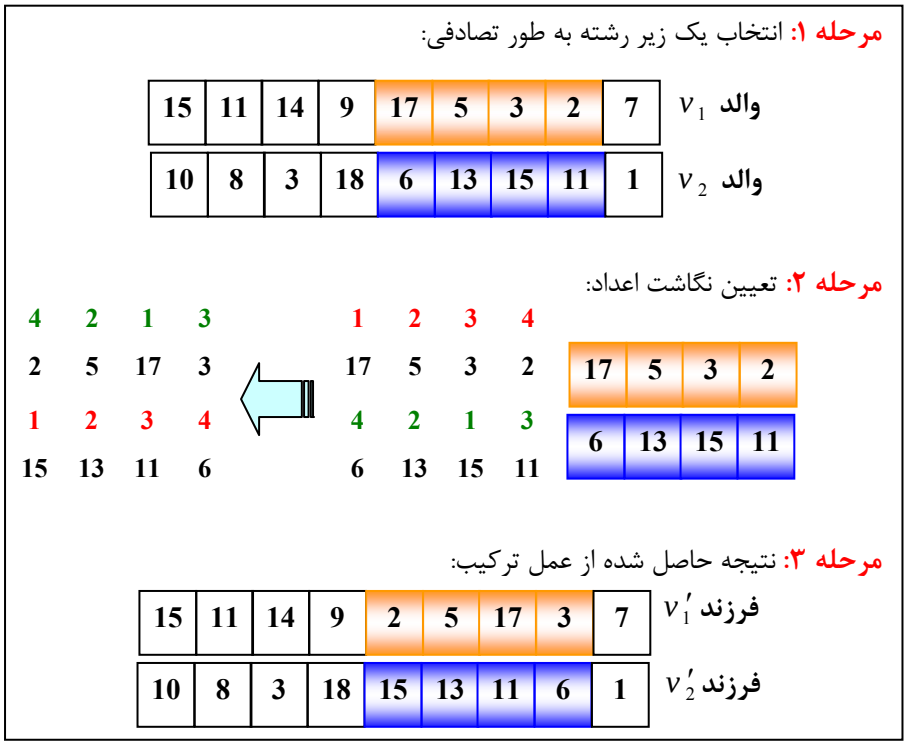
---

[۱] Weight Mapping Crossover

Weight Mapping Crossover Algorithm **(WMX)**

1. **Begin**
2. input: parent: $v1$, $v2$, number of tasks n
3. output: offspring: v_1', v_2'
4. $s \leftarrow$ random$[1, n-1]$;
5. $t \leftarrow$ random$[s+1, n]$;
6. $l \leftarrow t - s$;
7. for $(i = 1;\ i<=l; i++)$
8. $s_1[i] \leftarrow v_1[s+i]$;
9. $s_2[i] \leftarrow v_2[s+i]$;
10. $s_1[.] \leftarrow$ sorting$(s1[.])$;
11. $s_2[.] \leftarrow$ sorting$(s2)[.])$;
12. $v_1' \leftarrow v_1[1 : s-1] // v_2[s : t] // v_1[t+1 : n]$;
13. $v_2' \leftarrow v_2[1 : s-1] // v_1[s : t] // v_2[t+1 : n]$;
14. for $(i = 1;\ i<=l;\ i++)$
15. for $(j = 1;\ j<=l;\ j++)$
16. if $(v_1'[s+i] = s_2[j])$
17. $v_1'[s+i] \leftarrow s_1[j]$;
18. for $(j = 1;\ j<=l;\ j++)$
19. if $(v_2'[s+i] = s_1[j])$
20. $v_2'[s+i] \leftarrow s_2[j]$;
21. output: offspring: v_1', v_2' ;
22. **End.**

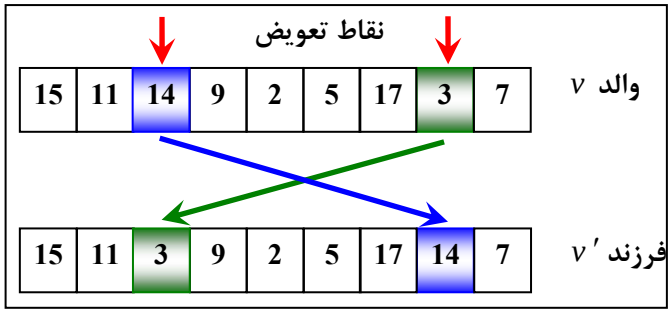
شکل الف-۱۳: الگوریتم WMX [۲].



شکل الف-۱۴: مثالی از ترکیب کروموزوم‌ها با استفاده از الگوریتم WMX [۲].

الف-۳-۴-۲ عمل‌گر جهش

در این الگوریتم برای انجام عمل جهش، ابتدا دو نقطه به طور تصادفی انتخاب شده و سپس محتویات آن‌ها با هم عوض می‌شود. شکل الف-۱۵ نحوه انجام عمل جهش را نمایش می‌دهد. زیر روال مربوط به عمل جهش نیز در الگوریتم شکل الف-۱۶ آورده شده است.



شکل الف-۱۵: نحوه انجام عمل جهش [۲].

Swap Mutation Algorithm
1. **Begin**
2. input: chromosome v , l
3. output: chromosome v'
4. $i \leftarrow$ random$[1 : l-1]$;
5. $j \leftarrow$ random$[i+1 : l]$;
6. $v' \leftarrow v[1 : i-1]\,//v[j]//v[i+1 : j-1]$
7. $//v[i]//v[j+1 : l]$;
8. output: offspring v' ;
9. **End.**

شکل الف-۱۶: زیر روال عمل جهش [۲].

الف-۳-۴-۳ عمل‌گر انتخاب

در این الگوریتم از روش چرخ رولت برای تولید جمعیت نسل جدید استفاده شده است، در این روش در ابتدا برازندگی هر کدام از کروموزوم‌ها محاسبه شده و در یک چرخ گردان قرار داده می‌شود. انتخاب به روش چرخ گردان به صورت زیر می‌باشد:

۱. مقدار برازندگی $eval(v_k)$ برای هر کدام از کروموزوم v_k محاسبه می‌شود.

$$eval(v_k) = f(x),\ k = 1, \ldots, popsize$$

۲. برازندگی کلی برای جمعیت محاسبه می‌شود.

$$F = \sum_{k=1}^{popsize} eval(v_k)$$

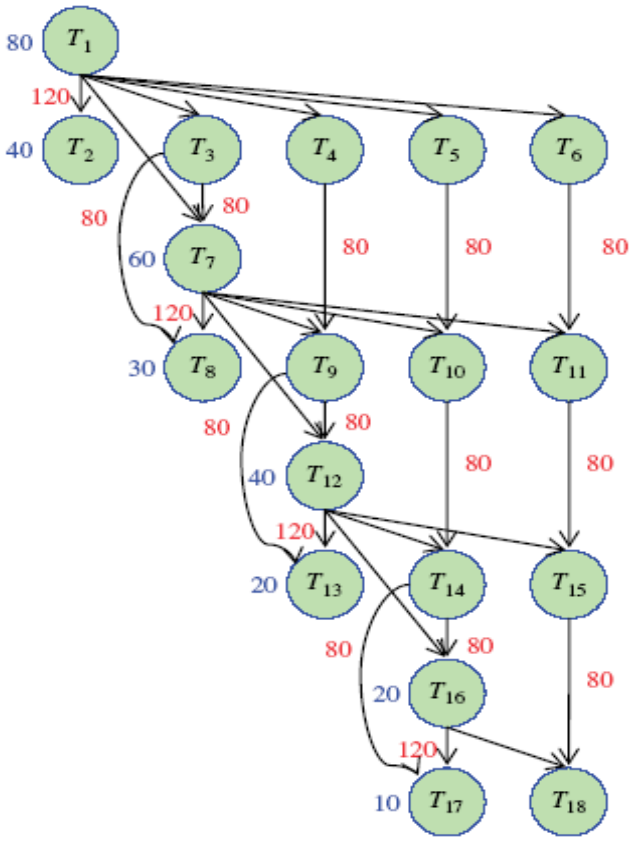
۳. احتمال انتخاب p_k برای هر کروموزوم v_k محاسبه می‌شود.

$$p_k = \frac{eval(v_k)}{F}, \quad k = 1, ..., popsize$$

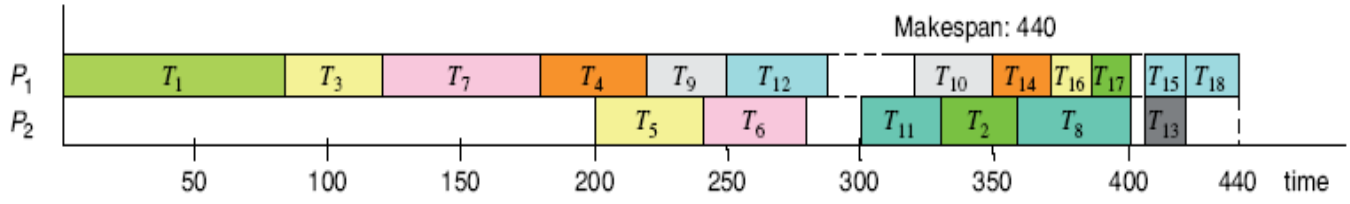
۴. احتمال دسته جمعی q_k برای هر کروموزوم v_k محاسبه می‌شود.

$$q_k = \sum_{j=1}^{k} p_j, \quad k = 1, ..., popsize$$

شکل الف-۱۸ نمودار گانت حاصل از الگوریتم زمان‌بندی را با استفاده از دو پردازنده برای DAG شکل الف-۱۷ را نمایش می‌دهد [۷۱].



شکل الف-۱۷: مثالی از یک DAG با تعداد ۱۸گره [۲].



شکل الف-۱۸: نمودارگانت حاصل از الگوریتم زمان‌بندی ژنتیک [۲].

پیوست ب

استفاده از خودکار یادگیر برای حل مساله زمان‌بندی چند پردازنده‌ای*

ب-۱ مقدمه

در این پیوست یک الگوریتم جدید با استفاده از خودکار یادگیر برای حل مساله زمان‌بندی چند پردازنده‌ای در سامانه‌های چند پردازنده‌ای معرفی می‌شود. همان طور که در پیوست اول مطرح شد، یک برنامه موازی با استفاده از گراف وظایف نمایش داده می‌شود و هدف از زمان‌بندی به حداقل رساندن زمان اجرای یک برنامه موازی بر روی سامانه‌های چند پردازنده‌ای است. برای زمان‌بندی یک گراف وظیفه با n گره در روی یک سیستم با p پردازنده $n! \times p^n$ زمان‌بندی مختلف وجود دارد. این مقدار زمانی درست است که هیچ وابستگی بین وظایف وجود نداشته باشد. در صورتی که وابستگی‌های گراف در نظر گرفته شود، تعداد زمان‌بندی‌های ممکن تقریبا برابر $n!$ است. در صورتی که از خودکار یادگیر برای حل مساله زمان‌بندی چند پردازنده‌ای استفاده شود، خودکار یادگیر باید حدود $n!$ اقدام داشته باشد. تعداد زیاد اقدام‌ها باعث کاهش سرعت هم‌گرایی می‌شود و به همین دلیل از خودکار مهاجرت اشیا (OMA[۱]) که به وسیله اومن[۲] و ما[۳] پیشنهاد شده است، استفاده شده است. در این پیوست در بخش ب-۲ خودکار یادگیر برای حل مساله زمان‌بندی چند پردازنده‌ای ارئه خواهد شد. در بخش ب-۳ الگوریتم جدید جهت زمان‌بندی چند پردازنده‌ای برای سامانه‌های چند پردازنده‌ای با استفاده از خودکار مهاجرت اشیا معرفی خواهد شد و در نهایت در بخش ب-۴ پیچیدگی زمانی الگوریتم محاسبه خواهد شد [۲، ۷۴، ۷۵].

---

* مطالب این پیوست برگرفته شده از مرجع [۲] می‌باشد.

۱ Object Migrating Automata

۲ Oommen

۳ Ma

ب-۲ خودکار یادگیر برای حل مساله زمان‌بندی چند پردازنده‌ای

خودکار یادگیر برای حل مساله زمان‌بندی چند پردازنده‌ای را می‌توان به صورت شش‌تایی $< \underline{V}, \underline{\alpha}, \underline{\varphi}, \underline{\beta}, \underline{F}, \underline{G} >$ نشان داد که در آن:

۱. $\underline{V} = \{V_1, V_2, ..., V_N\}$ مجموعه گره‌های گراف وظیفه $G = (V, E)$ می‌باشد. گره‌های گراف روی وضعیت‌های مختلف خودکار حرکت می‌کنند و با حرکت خود زمان‌بندی‌های مختلفی را ایجاد می‌کنند.

۲. $\underline{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_k\}$ مجموعه اقدام‌های مجاز برای خودکار یادگیر است. این خودکار k اقدام دارد (تعداد اقدام‌های این خودکار با تعداد گره‌های گراف وظیفه برابر است).

۳. $\underline{\varphi} = \{\varphi_1, \varphi_2, ..., \varphi_{kN}\}$ مجموعه وضعیت‌ها و N عمق حافظه برای خودکار می‌باشد. مجموعه وضعیت‌های این خودکار به k زیر مجموعه $\{\varphi_1, \varphi_2, ..., \varphi_N\}$ و $\{\varphi_{N+1}, \varphi_{N+2}, ..., \varphi_{2N}\}$ و... $\{\varphi_{(k-1)N+1}, \varphi_{(k-1)N+2}, ..., \varphi_{kN}\}$ افراز می‌شود و گره‌های گراف براساس این که در کدام وضعیت قرار داشته باشند دسته بندی می‌گردند. در مجموعه وضعیت‌های اقدام j ، به وضعیت $\varphi_{(j-1)N+1}$ وضعیت داخلی و به وضعیت φ_{jN} وضعیت مرزی گفته می‌شود. گرهی که در وضعیت $\varphi_{(j-1)N+1}$ قرار دارد گره با اهمیت بیشتر و گرهی که در وضعیت φ_{jN} گره با اهمیت کمتر نامیده می‌شود.

۴. $\underline{\beta} = \{0, 1\}$ مجموعه ورودی هر خودکار می‌باشد. در این مجموعه 1 شکست و 0 موفقیت را نشان می‌دهد.

۵. $\underline{F} : \underline{\varphi} \times \beta \rightarrow \underline{\varphi}$ تابع نگاشت وضعیت‌ها می‌باشد. این تابع از روی وضعیت فعلی و ورودی خودکار وضعیت بعدی آن را تولید می‌نماید. این تابع چگونگی گردش گره‌های گراف وظیفه را در وضعیت‌های خودکار مشخص می‌کند. این تابع برای خودکارهای مختلف، متفاوت می‌باشد که شرح آن در بخش بعد خواهد آمد.

۶. $G : \underline{\varphi} \rightarrow \underline{\alpha}$ تابع نگاشت خروجی می‌باشد. این تابع تصمیم می‌گیرد که به ازای هر وضعیت خودکار چه اقدامی را انجام می‌دهد.

ب-۳ زمان‌بندی چند پردازنده‌ای با استفاده از خودکار یادگیر

در بین الگوریتم‌های غیرقطعی ارایه شده برای حل مساله زمان‌بندی چند پردازنده‌ای، الگوریتم‌های ژنتیک نتایج بهتری را در مقابل بقیه الگوریتم‌های غیرقطعی نشان داده‌اند. الگوریتم‌های ژنتیک با یک جمعیت اولیه تصادفی کار خود را آغاز کرده و در جریان سیر تکاملی جمعیت بهینه شده و نسل‌های جدیدی ایجاد می‌شوند. در این الگوریتم تولید نسل زمانی خاتمه می‌گیرد که دیگر با ادامه تولید نسل جواب بهینه نشود. الگوریتم‌های ژنتیک سعی در پیدا کردن کروموزوم بهینه در بین جمعیت‌ها هستند و به محل قرار گیری ژن‌ها در کروموزوم‌ها توجه داده نمی‌شود و محل قرار گیری ژن‌ها در تمام کروموزوم‌ها به صورت تصادفی می‌باشد. اگر بتوان محل بهینه هر ژن در هر کروموزوم را مشخص کرد، می‌توان در تعداد نسل‌های کمتری به جواب نزدیک به بهینه دست یافت. در این بخش سعی می‌شود تا با استفاده از خودکار مهاجرت اشیا محل بهینه ژن‌ها برای هر کروموزوم مشخص شود. در خودکار مهاجرت اشیا، برای هر کروموزوم یک خودکار مهاجرت اشیا در نظر گرفته می‌شود. با سیر فرآیند یادگیری انتظار می‌رود که خودکار مهاجرت اشیا در تعداد تکرار کمتری به جواب بهینه نزدیک شود.

در این بخش برای زمان‌بندی چند پردازنده‌ای در سامانه‌های چند پردازنده‌ای، از خودکار مهاجرت اشیا مبتنی بر خودکارِ تستلین استفاده می‌شود. برای یک گراف وظیفه چندین خودکار می‌تواند وجود داشته باشد. هر خودکار نمایش دهنده یک زمان‌بندی می‌باشد. در زیر مراحل الگوریتم توضیح داده شده است.

ب-۳-۱ تولید جمعیت اولیه

در ابتدا به تعداد p خودکار به طور تصادفی برای گراف وظیفه مورد نظر ایجاد می‌شود. تعداد اقدام‌های هر خودکار برابر با تعداد گره‌های گراف وظیفه است. پس از تولید p خودکار به طور تصادفی، تمام وظایف با استفاده از الگوریتم شکل ب-۱ به اقدام‌های خودکارها انتساب داده می‌شوند.

```
Assign Task's to State's Algorithm
1. Begin
2. input: automata L, number of automata p, task t, number of tasks k, automata Action A.
3. output: p automata.
4.   for( j=1; j<=p; j++)
5.         for (i=1; i<=k; i++)
6.                   assign  t_i  to  L_j .A(i)
7. End.
```

شکل ب-۱: تولید جمعیت اولیه [۷۴].

پس از تولید p خودکار، به تمام اقدام‌های خودکارهای تولید شده یک عدد تصادفی تخصیص داده می‌شود. اعداد تصادفی تخصیص داده شده به اقدام‌ها نمایش دهنده دو مفهوم زیر می‌باشند:

الف. اولویت اجرای وظیفه‌ای که به آن اقدام انتساب داده شده است.

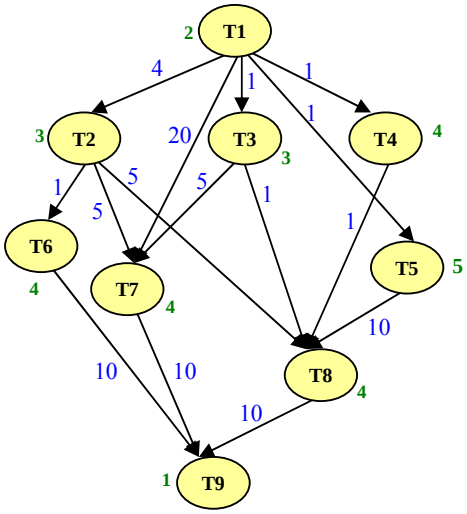
ب. پردازنده اجرا کننده آن وظیفه.

پس از تخصیص اعداد تصادفی به اقدام‌های خودکار به تعداد $k/2$ بار مقادیر اقدام‌ها با هم‌دیگر عوض می‌شوند. الگوریتم شکل ب-۲ نحوه تخصیص اعداد تصادفی به اقدام‌های خودکار را نمایش می‌دهد.

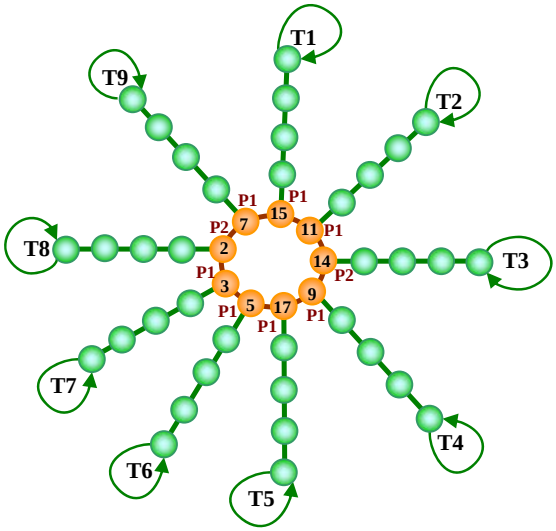
```
Assign Task's to Processors Algorithm
1. Begin
2. input: number of processors m, number of tasks k
3. output: Automata v(j )
4.          for (j = 1; j<=k; j++){
5.                  v(j ) ← m* j -random[0,m − 1];
6.          }\\end of for
7.          for (i = 1; i<= ⌊k/2⌋;i++){
8.                  j ← random[1, k];
9.                  l ← random[1, k];
10.                 if (j ≠ l)
11.                    swap (v(j ), v(l))
12.                 }\\end of for
13.         output: the  Automata v(j );
14.End.
```

شکل ب-۲: نحوه تخصیص وظایف به پردازنده‌ها [۷۴].

حال برای مثال گراف نمایش داده شده در شکل ب-۳ در یک سامانه چندپردازنده‌ای که دارای 2 واحد پردازشی می‌باشد به اجرا در آورده می‌شود. در شکل ب-۴ مشاهده می‌شود که هرکدام از وظایف به ترتیب به اقدام‌های خودکار تخصیص داده شده‌اند. چون در این سامانه دو پردازنده وجود دارد بنابراین اعداد فرد مشخص کننده پردازنده p1 و اعداد زوج مشخص کننده p2 می باشند. در صورتی‌که سامانه بیشتر از دو پردازنده داشته باشد در این صورت شماره پردازنده عبارت است از باقی‌مانده عدد تخصیص داده شده به اقدام‌های خودکار، به تعداد کل پردازنده‌ها.

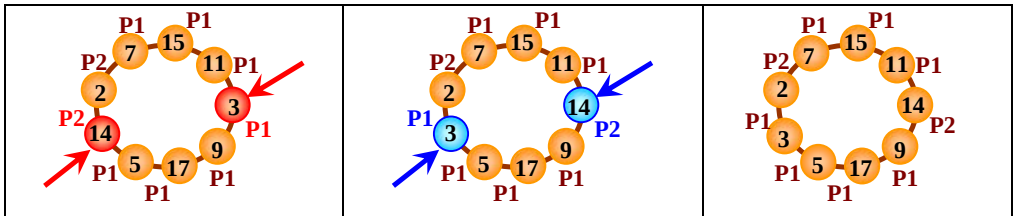


شکل ب-۳: مثال از گراف وظیفه [۷۴].



شکل ب-۴: خودکار تصادفی برای گراف وظیفه شکل ب-۳ [۷۴].

شکل ب-۵ نحوه تعویض مقادیر تخصیص داده شده به اقدام‌ها را نمایش می‌دهد.



(الف) وضعیت اقدام‌های خودکار پس از تخصیص مقادیر تصادفی

(ب) انتخاب دو اقدام تصادفی

(پ) خروجی نهایی برای خودکار پس از تعویض مقادیر اقدام‌های انتخاب شده

شکل ب-۵: مثالی از نحوه تعویض مقادیر دو اقدام تصادفی [۷۴].

ب-۳-۲ نحوه اجرای وظایف بر روی پردازنده‌ها

موقع اجرا شدن یک برنامه بر روی پردازنده‌های موازی باید وابستگی داده‌ای بین وظایف نیز در نظر گرفته شود. در واقع یک وظیفه نمی‌تواند اجرا شود مگر این که در گذشته تمام وظیفه‌های والدش به اجرا در آمده باشند. در این بخش نحوه اجرای وظایف، بر روی پردازنده‌ها توضیح داده خواهد شد. هر وظیفه با توجه به خودکار شکل ب-۴ در پردازنده مربوطه خود اجرا می‌شود. روال کار بدین صورت است که از بین تمام وظایف آماده وظیفه‌ای به اجرا در می‌آید که اولویت آن از اولویت بقیه وظایف بیشتر باشد. وظیفه آماده وظیفه‌ای است که تمام وظایف والد آن اجرا شده باشد.

برای مثال با در نظر گرفتن خودکار شکل ب-۴ و گراف وظیفه شکل ب-۳ در مرحله اول فقط وظیفه T_1 در حال آماده می‌باشد، بنابراین در مرحله اول T_1 به اجرا در خواهد آمد. پس از اجرا شدن وظیفه T_1، وظایف T_2، T_3، T_4 و T_5 به حالت آماده در می‌آیند، در این مرحله اولویت وظیفه T_5 از بقیه وظایف آماده بیشتر است، بنابراین وظیفه T_5 به اجرا در می‌آید. به همین ترتیب تمام وظایف بر روی پردازنده‌های مربوط به خود به اجرا در می‌آیند. الگوریتم شکل ب-۶ ترتیب اجرا شدن وظایف بر روی پردازنده‌های مربوطه را مشخص می‌کند. با توجه به خودکار شکل ب-۴ وظایف $T_1, T_5, T_2, T_4, T_6, T_7, T_9$ بر روی پردازنده p1 و وظایف T_3, T_8 بر روی پردازنده p2 به اجرا در می‌آیند.

Task Execution Priority Algorithm
1. **Begin**
2. input: number of tasks *k*, *Automata v(j)*, the set of nodes $\overline{S}$
3. output: task sequence *TS*
3. $\overline{S} \leftarrow \varnothing$, $TS \leftarrow \varnothing$;
4. $n \leftarrow 0, j \leftarrow 0$;
5. while $(j \leq k)$ {
6. $\overline{S} \leftarrow \overline{S} \cup suc_j$;
7. $j* \leftarrow \arg\max\{v(j) | j \in \overline{S}\}$;
8. $\overline{S} \leftarrow \overline{S} \setminus j*$;
9. $TS \leftarrow TS \cup j*$;
10. $j \leftarrow j*$;
11. }\\end of while
12. output: task sequence *TS*;
13. **End.**

الگوریتم ب-۶: الگوریتم مشخص کردن ترتیب اجرا وظایف [۷۴].

J	$\bar{S}$	$v\ (j)$	J^*	TS
0	{1}	v (1) = 15	1	S = {1}
2	{2, 3, 4, 5}	v (2) = 11, v (3) = 14, (4) = 9, v (5) = 17	5	S = {1, 5}
1	{2, 3, 4}	v (2) = 11, v (3) = 14, (4) = 9	3	S = {1, 5, 3}
3	{2, 4}	v (2) = 11, (4) = 9	2	S = {1, 5, 3, 2}
5	{4, 6, 7}	v (4) = 9, v (6) = 5, v (7) = 3	4	S = {1, 5, 3, 2, 4}
4	{6, 7, 8}	v (6) = 5, v (7) = 3, v (8) = 2	6	S = {1, 5, 3, 2, 4, 6}
7	{7, 8}	v (7) = 3, v (8) = 2	7	S = {1, 5, 3, 2, 4, 6, 7}
9	{8}	v (8) = 2	8	S = {1, 5, 3, 2, 4, 6, 7, 8}
6	{9}	v (9) = 7	9	S = {1, 5, 3, 2, 4, 6, 7, 8, 9}

شکل ب-۷: ترتیب اجرای وظایف گراف شکل ب-۴ [۷۴].

حال وظایف باید به همان ترتیبی که در جدول شکل ب-۷ نمایش داده شده است به اجرا در آیند. همان طور که در الگوریتم شکل ب-۸ مشاهده می‌شود خروجی حاصل از این الگوریتم یک زمان‌بند می‌باشد.

Task Execution Algorithm

1. **Begin**
2. input: processing time p_k, task sequence *TS*, Automata *v(j)*, the communication delay τ_{jk}
3. output: makespan *f*, schedule *S*
4. $Pi \leftarrow 0,\ i = 1, 2, \ldots, m;$
5. $t_k \leftarrow 0,\ k = 1, 2, \ldots, n;$
6. $s \leftarrow 0;$
7. for ($j = 1$;j<= n; j++){
8. $s \leftarrow TS_i$;
9. $Pi \leftarrow v(s)\%m;$
10. if ($p_i = 0$) $p_i \leftarrow m;$
11. if (assigned task $T_j < T_k$)
12. if ($p_i \neq p_l$)
13. $e_k \leftarrow \max\{t_j \mid j \in \text{pre}(k)\} + \tau_{jk}$;
14. $t_k \leftarrow e_k + p_k$;
15. $S \leftarrow S \cup S_k \{ p_i ; e_k + p_k \};$
16. else $e_k \leftarrow \max\{t_j \mid j \in pre(k)\};$
17. $t_k \leftarrow e_k + p_k$;
18. $S \leftarrow S \cup S_k \{ p_i ; e_k + p_k \};$
19. else $e_k \leftarrow \max\{t_j \mid j \in pre(k)\};$
20. $t_k \leftarrow e_k + p_k$;
21. $S \leftarrow S \cup S_k \{ p_i ; e_k + p_k \};$
22. }\\end of for
23. output: schedule *S*
24. makespan $f \leftarrow \max\{t_k, k = 1, 2, \ldots, n\};$
25. **End**.

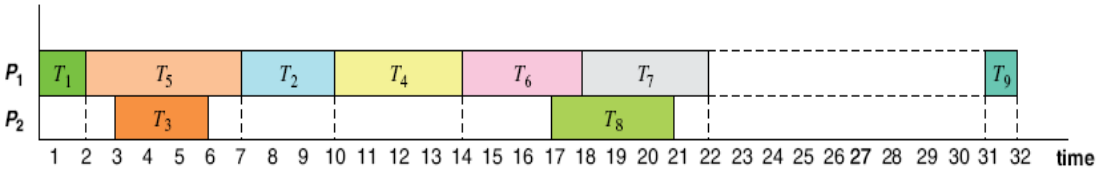
شکل ب-۸: نحوه اجرای وظایف [۷۴].

جدول شکل ب-۹ ترتیب اجرای وظایف بر روی پردازنده‌ها را قدم به قدم نمایش می‌دهد.

J^*	S	P_i	$t_j = e_j + p_j$
J^*	S	P_i	$t_j = e_j + p_j$
1	P1 = {1}, p2 = {}	1	$t_1 = 0 + 2 = 2$
5	P1 = {1, 5}, p2 = {}	1	$t_5 = 2 + 5 = 7$
3	P1 = {1, 5}, p2 = {3}	2	$t_3 = 3 + 3 = 6$
2	P1 = {1, 5, 2}, p2 = {3}	1	$t_2 = 7 + 3 = 10$
4	P1 = {1, 5, 2, 4}, p2 = {3}	1	$t_4 = 10 + 4 = 14$
6	P1 = {1, 5, 2, 4, 6}, p2 = {3}	1	$t_6 = 14 + 4 = 18$
7	P1 = {1, 5, 2, 4, 6, 7}, p2 = {3}	1	$t_7 = 18 + 4 = 22$
8	P1 = {1, 5, 2, 4, 6, 7}, p2 = {3,8}	2	$t_8 = 17 + 4 = 21$

شکل ب-۹: نمایش قدم به قدم ترتیب اجرای وظایف [۷۴].

پس از اجرای الگوریتم شکل ب-۸ برای خودکار نمایش داده شده در شکل ب-۴ نحوه اجرای وظایف به صورت نمودار گانت نمایش داده شده در شکل ب-۱۰ می‌باشد.



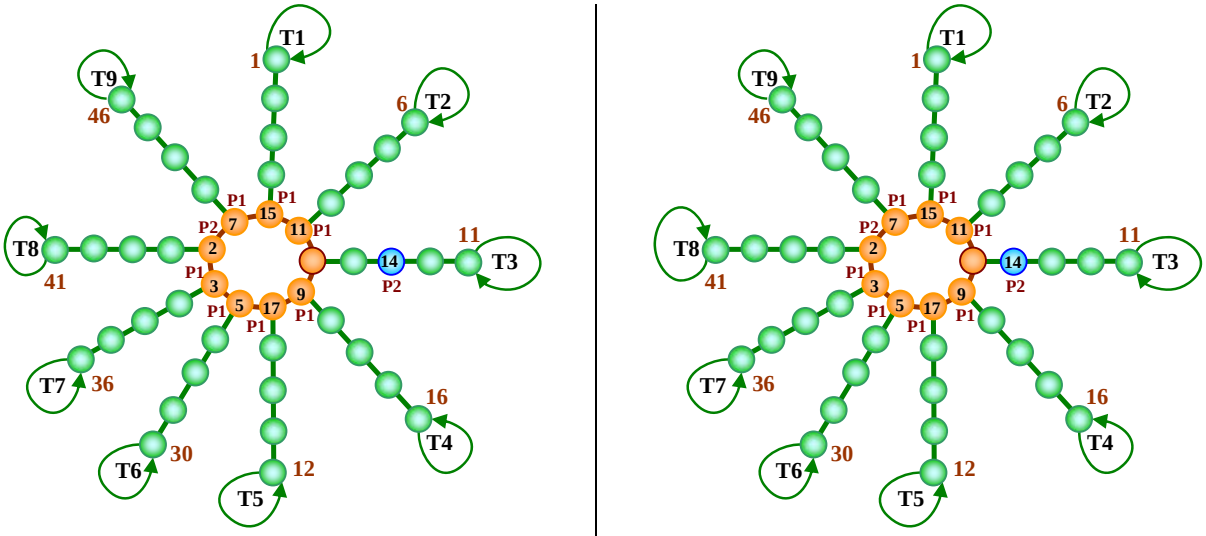
شکل ب-۱۰: نمایش اجرای وظایف گراف شکل ب-۳ با استفاده از خودکار شکل ب-۴ [۷۴].

ب-۳-۳ عمل‌گر جریمه وپاداش

در هر یک از خودکارها یک اقدام به صورت تصادفی انتخاب شده و آن اقدام مورد عمل پاداش یا جریمه قرار می‌گیرد. در اثر پاداش دادن یا جریمه کردن یک اقدام، وضعیت آن اقدام در مجموعه وضعیت‌های اقدام مربوط، تغییر می‌کند. اگر یک اقدام در وضعیت مرزی قرار داشته باشد، جریمه شدن آن باعث تغییر اقدام آن و در نتیجه باعث ایجاد یک زمان‌بندی جدیدی می‌شود. عمل‌گر جریمه و پاداش با توجه به نوع خودکار یادگیر متفاوت خواهد بود. در این پیوست از خودکار مهاجرت اشیا مبتنی بر خودکارِ تستلین استفاده شده است. الگوریتم شکل ب-۱۱ نحوه انجام عمل پاداش را نمایش می‌دهد.

```
Reward Algorithm
1. Begin
2.    if ((L.s-1)%n<>0) then
3.          L.s--;
4. End.
```

شکل ب-۱۱: نحوه انجام عمل پاداش یک خودکار [۷۴].



(الف) وضعیت خودکار قبل از پاداش دهی به وظیفه t_3

(ب) وضعیت خودکار بعد از پاداش دهی به وظیفه t_3

شکل ب-۱۲: نحوه پاداش دهی به وظیفه t_3 [۷۴].

عمل پاداش زمانی صورت می‌گیرد که میزان برازندگی یک وظیفه از مقدار آستانه کوچک‌تر باشد.

میزان برازندگی وظیفه t_i عبارت است از: x / y.

x: عبارت است از مجموع هزینه ارتباطی تمام گره‌های والد و فرزند گره t_i به طوری که $p_{t_i} \neq p_{t_j}$.

$$\left[\sum c(t_i, t_j) \quad if \quad (p_{t_i} \neq p_{t_j})\right]$$

y: عبارت است از مجموع هزینه ارتباطی تمام گره‌های فرزند و والد گره t_i. $\sum c(t_i, t_j)$

p_{t_i} : پردازنده‌ای که وظیفه t_i در آن اجرا می‌شود.

p_{t_j} : پردازنده‌ای که وظیفه t_j در آن اجرا می‌شود.

$c(t_i, t_j)$:عبارت است از هزینه ارتباطی بین وظیفه t_i و t_j .

مقدار آستانه برابر است با $T / Ntasks$.

T: عبارت است از تعداد وظایف وابسته به وظیفه t_i که در پردازنده‌ای اجرا می‌شوند که وظیفه t_i در آن اجرا می‌شود.

Ntasks: عبارت است از تعداد کل وظایف گراف.

هرچه میزان برازندگی وظیفه t_i به صفر میل کند به همان اندازه هزینه ارتباطی بین پردازنده‌ها به صفر میل می‌کند، پس اگر میزان برازندگی وظیفه t_i برابر صفر باشد دلیل بر این است که تمام وظایف وابسته t_i بر روی یک پردازنده اجرا می شوند. T با x رابطه مستقیمی دارد. به طوری که با افزایش مقدار T، مقدار x کم می‌شود و با کاهش مقدار T، مقدار x افزایش داده می‌شود.

(ب-۱)

عمل جریمه $x / y \geq T / Ntasks$

(ب-۲)

عمل پاداش $x / y < T / Ntasks$

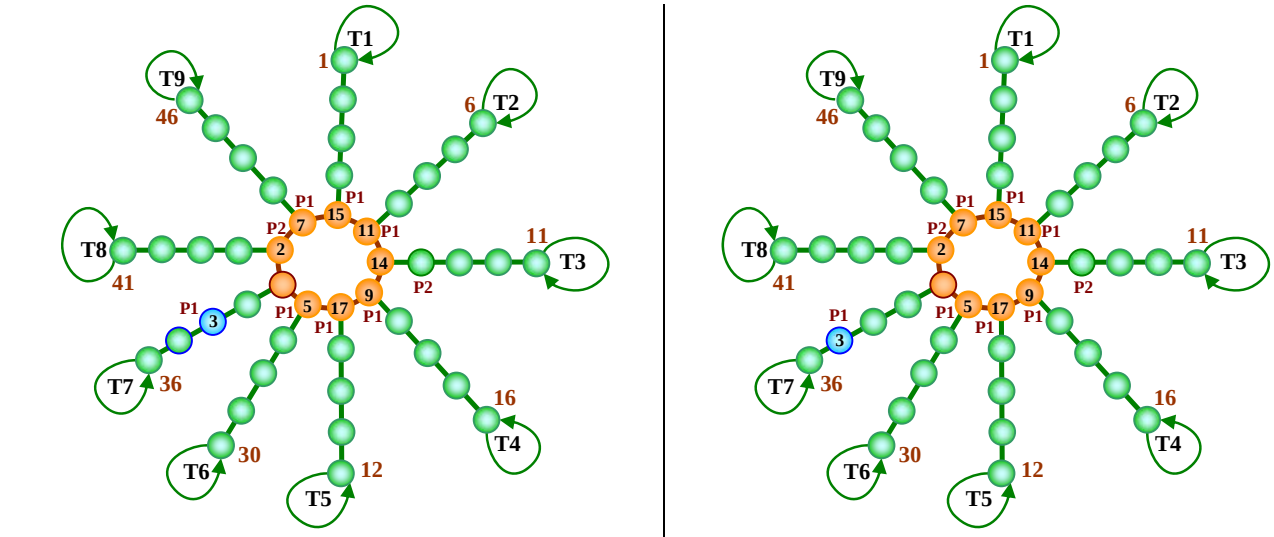
همان طور که در شکل ب-۴ مشاهده می‌شود این خودکار شامل ۹ وظیفه می‌باشد و شکل ب-۹ نحوه اجرای وظایف بر روی پردازنده‌ها را نمایش می‌دهد. موقع پاداش دادن به یک وظیفه عدد انتساب داده شده به اقدام مربوط تغییر وضعیت می‌دهد. حال اگر اقدام انتخاب شده وظیفه t_3 باشد نحوه پاداش دهی به صورت خودکار شکل ب-۱۲ می‌شود. در صورتی عدد انتساب داده شده به یک وظیفه، در وضعیت نهایی (برای مثال در وضعیت ۶ خودکار) خود قرار داشته باشد، در این صورت با اعمال عمل پاداش هیچ عملی صورت نمی‌گیرد.

در صورتی که میزان برازندگی یک وظیفه بزرگ‌تر یا مساوی مقدار آستانه باشد در این صورت راس وظیفه مورد نظر جریمه می‌شود. الگوریتم شکل ب-۱۳ نحوه جریمه شدن یک راس را نمایش می‌دهد. موقع جریمه کردن یک راس دو حالت ممکن صورت گیرد:

الف. راس مورد نظر در وضعیتی غیر از وضعیت مرزی قرار داشته باشد. در این صورت جریمه کردن این راس سبب کم اهمیت شدن این راس می‌شود. برای مثال نحوه جریمه شدن راس مربوط به وظیفه t_7 را می‌توان در شکل ب-۱۴ مشاهده نمود.

```
Penalizing Algorithm
1. Begin
2.    do{
3.        for {u = 1; u<= n;u++){
4.            if ((L.s(U)) % N <> 0) then
5.                L.s(U)++;
6.        }\\end of for
7.    }while (at least one node appears in the boundary state)
8.    best sequence makespan;
9.    for (U = 1; U<= n; U++){
10.       create scheduling LA′ from LA by swapping u and U
11.         if  (Lenght(makespan (LA′) <best makespan Length)
12.             best makespan Lenght = Lenght(makespan(LA′);
13.             bestNode = U;
14.         }\\end of if
15.   }\\end of for
16.   L.S(bestNode) = L.A(bestNode)*N;
17.   L.S(u) = L.A(u)*N;
18.   Swap(L.S(u),L.S(bestNode));
19. End
```

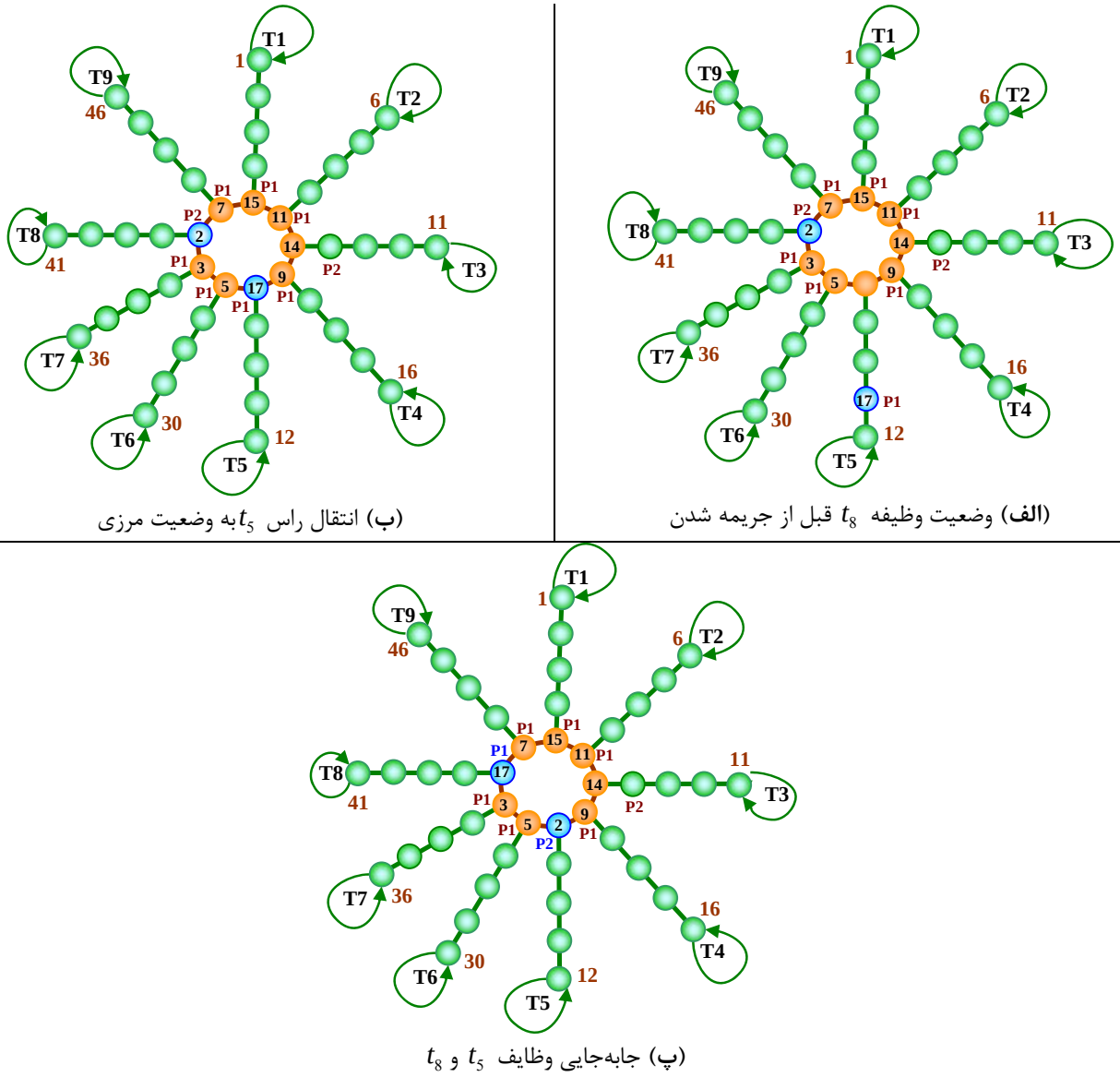
شکل ب-۱۳: روال جریمه شدن یک راس [۷۴].



(الف) وضعیت خودکار قبل از جریمه شدن وظیفه t_7 **(ب)** وضعیت خودکار بعد از جریمه شدن وظیفه t_7

شکل ب-۱۴: نحوه جریمه شدن وظیفه t_7 [۷۴].

ب. راس مورد نظر در وضعیت مرزی قرار داشته باشد. در این حالت راسی از گراف پیدا می‌شود به طوری که با تعویض پردازنده‌ها (اعداد انتساب داده شده به راس‌ها) بیشترین کاهش در مقدار FT داشته باشد. حال اگر راس پیدا شده در وضعیت مرزی قرار داشته باشد جای دو راس با هم‌دیگر عوض می‌شود و در غیر این صورت ابتدا راس پیدا شده به وضعیت مرزی خود انتقال داده می‌شود و سپس جابه‌جایی صورت می‌گیرد. در شکل ب-۱۵ نحوه جریمه شدن وظیفه t_8 را نمایش می‌دهد.



(الف) وضعیت وظیفه t_8 قبل از جریمه شدن

(ب) انتقال راس t_5 به وضعیت مرزی

(پ) جابه‌جایی وظایف t_5 و t_8

شکل ب-۱۵: نحوه جریمه شدن وظیفه t_8 [۷۴].

ب-۴ پیچیدگی زمانی الگوریتم

در این بخش پیچیدگی زمانی الگوریتم معرفی شده در بخش ب-۳ محاسبه می‌شود. برای محاسبه پیچیدگی زمانی الگوریتم، لازم است تا در ابتدا مفروضات زیر در نظر گرفته شود:

n: تعداد جمعیت اولیه

i: تعداد تکرار (تعداد مراحل یادگیری)

p: احتمال پاداش گرفتن یک اقدام

1-p: احتمال جریمه شدن یک اقدام

v: تعداد کل گره‌های گراف وظیفه (تعداد اقدام‌های خودکارها)

e: تعداد کل یال‌های گراف وظیفه
P_r: تعداد پردازنده‌های سامانه

با در نظر گرفتن مفروضات بالا پیچیدگی الگوریتم به صورت زیر می‌باشد:

$$i\ (n\ [\ p\ (\text{پاداش}) + 1\text{-}p\ (\text{جریمه})\ ]\ )$$

حال پیچیدگی زمانی پاداش‌دهی و جریمه نمودن یک اقدام محاسبه می‌شود. موقع پاداش گرفتن یک اقدام فقط وضعیت آن اقدام تغییر می‌کند، یعنی وضعیت اقدام به وضعیت داخلی خود حرکت می‌کند و در صورتی که در داخلی‌ترین وضعیت قرار داشته باشد، هیچ تغییری در وضعیت آن اقدام صورت نمی‌گیرد. بنابراین زمان پاداش گرفتن یک اقدام عبارت است از O (1). ولی برای جریمه شدن یک اقدام لازم است تا به تعداد اقدام‌های خودکار (گره‌های گراف وظیفه) اقدام‌های خودکار با هم‌دیگر جابه‌جا شوند و در هر بار جابه‌جایی زمان FT برای هر خودکار محاسبه می‌شود و در نهایت خودکار وضعیتی را برای خود انتخاب می‌کند که زمان FT آن از بقیه کمتر باشد. بنابراین برای محاسبه پیچیدگی زمانی جریمه شدن یک گره خواهیم داشت:

$$V.FT$$

اما برای به دست آوردن زمان FT در ابتدا نمودار گانت خودکار مورد نظررا به دست آورده و سپس زمان خاتمه آخرین وظیفه را محاسبه می‌کنیم. بنابراین برای محاسبه زمان FT خواهیم داشت:

$$FT = (v + v + e.P_r\ )$$

در نهایت پیچیدگی الگوریتم عبارت است از:

$$i.[n\ (p\ (1) + (1 - p)\ (v\ (v + v + e.P_r\ )))]\ \ =>$$

در بدترین حالت تعداد یال‌های گراف برابر $v\ (v - 1)$ می‌باشد. بنابراین خواهیم داشت:

$$i.[n\ (p\ (1) + (1 - p)\ (v\ (v + v + v^2.P_r)))]\ \ =>$$

$$i.[n\ (p\ (1) + (1 - p)\ (\ v^3.P_r\ ))]\ \ =>$$

$$i.[n\ (\ v^3.P_r\ )]$$

چون P_r عدد ثابت است بنابر این داریم:

$$i.[n\ (\ v^3)]$$

از آن‌جایی که i و n نیز اعداد ثابت می‌باشند لذا تاثیری در پیچیدگی زمانی ندارند. در نهایت پیچیدگی الگوریتم برابر است با $O\ (v^3)$

مراجع

[۱] میر مهدی سید اسفهلان، «کاربرد الگوریتم‌های تکاملی برای تحلیل مسایل آنتن و مایکروویو»، سمینار کارشناسی ارشد ، دانشگاه علم و صنعت- دانشکده‌ی مهندسی برق- گروه مهندسی مخابرات، ۱۳۸۷.

[۲] حبیب مطیع قادر، «کاربرد آتوماتای یادگیر در زمان‌بندی ایستای گراف وظایف بر روی سکوهای کاری همگن و ناهمگن»، پایان نامه کارشناسی ارشد، دانشگاه آزاد اسلامی واحد شبستر- دانشکده فنی مهندسی کامپیوتر، مهر ۱۳۸۷.

[۳] باقر زارعی، «پردازش تکاملی»، جزوه آموزشی، دانشگاه آزاد اسلامی واحد شبستر- دانشکده کامپیوتر، ۱۳۸۷.

[۴] ناصر لطفی، «زمان‌بندی گراف وظایف برای پردازش موازی مبتنی بر پردازش تکاملی»، پایان نامه کارشناسی ارشد، دانشگاه آزاد اسلامی واحد نجف آباد- دانشکده کامپیوتر، ۱۳۸۴.

[5] R. L. Haupt and S. E. Haupt, "Practical Genetic Algorithms, 2nd Edition", John Wiley & Sons Inc, 2004.

[6] P. Pedregal, "Introduction to Optimization", Springer, New York Inc., 2004.

[7] S. B. L. Vandenberghe, "Convex Optimization", Cambridge University Press, 2004.

[8] E. K. P. Chong and S. H. Żak, "An Introduction to Optimization", John Wiley & Sons Inc, 2001.

[9] W. Sun and Y.Yuan, "Optimization Theory and Methods: Nonlinear Programming", Springer Science + Business Media, LLC Press, 2006.

[10] J. Nocedal and S. J. Wright, "Numerical Optimization, 2nd Edition", Springer Science + Business Media, LLC Press, 2006.

[11] A. Brabazon and M. O'Neill, "Biologically Inspired Algorithms for Financial Modeling", Springer-Verlag Berlin Heidelberg, 2006.

[12] C. L. Karr and L.M.Freeman, "Industrial Applications of genetic Algorithms", CRC Press, 1999.

[13] C. David, "An Interoduction to Genetic Algorithms for Scientists and Engineers", World Scientific, 1999.

[14] S. N. Sivanandam and S. N. Deepa, "Introduction to genetic algorithms" ,Springer-Verlag Berlin Heidelberg, 2008.

[15] B. Zarei, M. R. Meybodi and M. Abbaszadeh, "A Hybrid Method for Solving Traveling Salesman Problem", Computer and Information Science, ICIS 2007.

[16] S. Olariu and A.Y. Zomaya, "Handbook of Bioinspired Algorithms and Applications", Taylor & FrancisGroup, LLC Press, 2006.

[17] J. Gottlieb and G. R. Raidl, "Evolutionary Computation in Combinatorial Optimization", Springer-Verlag Berlin Heidelberg, 2006.

[18] F. Glover and G. A. Kochenberger, "Hanbook of Metaheuristics", Kluwer Academic Publishers, 2003.

[19] P. Siarry, "Metaheuristics for Hard Optimizations", Springer-Verlag Berlin Heidelberg, 2006.

[20] R. Mendes, "Population topologies and their influence in particle swarm performance", PhD Thesis, Department of Informatics, School of Engineering, University of Minho, 2004.

[21] R. Mendes, J. Kennedy and J. Neves, "Watch thy neighbor or how the swarm can learn form its environment"t, in Proceedings of IEEE Swarm Intelligence Symposium, pp. 88-94, 2003.

[22] A. P. Engelbrecht, "Computational Intelligence, 2nd Edition", John Wiley & Sons Ltd, 2007.

[23] A. Hoorfar, "Evolutionary Programming in Electromagnetic Optimization", IEEE Transactions on Antennas and Propagation, vol 55, no. 3, March 2007.

[24] A. Qing, "Electromagnetic Inverse Scattering of Multiple Two-Dimensional Perfectly Conducting Objects by the Differential Evolution Strategy", IEEE Transactions on Antennas and Propagation, vol 51, no. 6, June 2003.

[25] C. R. Reeves and J. E. Rowe, "Genetic Algorithms: Principles and Perspectives", Kluwer Academic Publishers, 2003.

[26] S. Alfonzetti, E. Dilettoso and N. Salerno, "Simulated Annealing With Restarts for the Optimization of Electromagnetic Devices", IEEE Transactions on Magnetics, vol 42, no. 4, April 2007.

[27] J. M. Fernandez, J. M. Gil and J. Zapata, "Ultrawideband Optimized Profile Monopole Antenna by Means of Simulated Annealing Algorithm and the Finite Element Method", IEEE Transactions on Antennas and Propagation, vol 55, no. 6, June 2007.

[28] M. Dorigo and L. M. Gambardella, "Ant Colony Optimization and Swarm Intelligence", Springer-Verlag Berlin Heidelberg, 2006.

[29] E. R. lglesias and O. Q. Teruel, "Linear Array Synthesis Using an Ant-Colony-Optimization-Based Algorithm", IEEE Transactions on Antennas and Propagation, vol 49, no. 2, April 2007.

[30] J. Kennedy and R. C. Eberhart, "Swarm Intelligence", Academic Press, 2001.

[31] N.Nedjah and L.M. Mourelle, "Swarm Intelligent Systems", Springer-Verlag Berlin Heidelberg, 2006.

[32] M. Dorigo and G. D. Caro, "The Ant Colony Optimization Metaheuristic", Iridia University, 1999.

[33] M. Dorigo and L. M. Gambardella, "Ant colonies for the traveling salesman problem", Bio-systems, no. 43, pp. 73-81, 1997.

[34] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem", IEEE Transaction on Evolutionary Computation, no. 1, pp. 53-66, 1997.

[35] L. M. Gambardella and M. Dorigo, "Ant-Q: A reinforcement learning approach to the traveling salesman problem", in Proceedings of 12th International Conference on Machine Learning, pp. 252-260, 1995.

[36] L. M. Gambardella and M. Dorigo, "Solving symmetric and asymmetric TSPs by ant colonies", in Proceedings of IEEE Conference on Evolutionary Computation, pp. 622-627, 1996.

[37] D. Corne, M. Dorigo and F. Glover, "New Ideas in Optimization", McGraw-Hill, 1999.

[38] L. M. Gambardella, E. D. Taillard and M. Dorigo, "Ant Colonies for the QAP", Technical Report, IDSIA, 1997.

[39] T. Stützle and H. Hoos, "Max-min ant system", Future Generation Computer Systems, no. 16, pp. 889-914, 2000.

[40] E. Bonabeau, M. Dorigo and G. Theraulaz, "Swarm Intelligence: From Natural to Artificial Systems", Oxford University Press, New York, 1999.

[41] B. Bullnheimer, R. Hartel and C. Strauss, "A new rank-based version of the ant system: A computational study, Central European Journal of Operations Research and Economics", no. 7, pp. 25-38, 1999.

[42] M. Dorigo, V. Maniezzo and A. Colorni, "The Ant System: Optimization by a colony of cooperating agents", IEEE Transactions on Systems, Man and Cybernetics, part B, no. 26, pp. 29-41, 1996.

[43] T. Stützle and H. Hoos, "Improvments on the Ant System: Introducing Max-Min Ant System", in Proceedings of Internation Conference on Artificial Neural Networks and Getenic Algorithms, Vienna, Springer-Verlag, 1997.

[44] B. Bullnheimer, R. Hartel and C. Strauss, "An improved ant system algorithm for the vehicle routing problem", Technical Report, University of Vienna, 1997.

[45] M. Reimann, K. Doerner and R. Hartel, "Analysing a unified ant system for the VRP and some of its variants", in Applications of Evolutionary Computing, Evo Workshops, Springer-Verlag Berlin Heidelberg, 2003.

[46] V. Maniezzo and A. Colorni, "The Ant System applied to a Quadratic Assignemnt Problem", IEEE Transactions on Knowledge and Data Engineering, vol 30, no. 5, pp 769-778, 1998.

[47] T. Stützle and H. Hoos, "Max-Min Ant System and local search for combinatorial optimization problems", in Proceedings of Metaheuristics International Conference, 1997.

[48] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization", in Proceedings of the 4th IEEE International Conference on Neural Networks, pp. 1942-1948, 1995.

[49] R. C. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory", in Proceedings of the 6th International Symposium on Micro Machine and Human Science, pp. 39-43, 1995.

[50] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer", in Proceedings of IEEE International Conference on Evolutionary Computation, pp. 69-73, 1998.

[51] S. Camazine and J. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz, E. Bonabeau, "Self-Organization in Biological Systems", Princeton University Press, 2000.

[52] Y. Shi and R. C. Eberhart, "Comparing inertia weights and constriction factors in Particle Swarm Optimization", in Proceedings of IEEE International Congress on Evolutionary Computation, pp. 84-88.

[53] Y. Shi and R. C. Eberhart, "Fuzzy Adaptive Particle Swarm Optimization", in Proceedings of IEEE International Conference on Evolutionary Computation, pp. 101-106, 2001.

[54] H. Liu and A. Abraham, "Fuzzy Turbulent Particle Swarm Optimization", in Proceedings of 5th IEEE International Conference on Hybrid Intelligent Systems, 2005.

[55] M. Clerc and J. Kennedy, "The Particle Swarm: Explosion, Stability and Convergence in multi-dimensional complex space", IEEE Transactions on Evolutionary Computation, vol 20, no. 6, pp. 58-73, 2002.

[56] J. Kennedy and R. Mendes, "Population structure and particle swarm performance", in Proceedings of IEEE International Conference on Evolutionary Computation, pp. 1671-1676, 2002.

[57] M. Donelli, R. Azaro, F. G. B. De Natale and A. Massa, "An Innovative Computational Approach Based on a Particle Swarm Strategy for Adaptive Phased-Arrays Control", IEEE Transactions on Antennas and Propagation, vol 54, no. 3, March 2006.

[58] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction", Cambridge, MIT Press, 1998.

[59] A. S. Poznyak and K. Najim, "Learning Automata and Stochastic optimization", Springer-verlaqg London Limited 1997.

[60] K. S. Narendra and M. A. L. Thathachar, "Learning Automata: An introduction", Prentice Hall, 1989.

[61] M. R. Meybodi and H. Beigy, "New Class of Learning Automata Based Scheme for Adaptation of Backpropagation Algorithm Parameters", in Proceeding of EUFIT-98, Sep. 7-10, Achen, Germany, pp. 339-344, 1998.

[62] B. J. Oommen and D. C. Y. Ma, "Deterministic Learning Automata Solution to the Keyboard Optimization Problem", IEEE Trans. On Computers, Vol. 37, No. 1, pp. 2-3, 1988.

[63] H. Beigy and M. R. Meybodi, "Optimization of Topology of neural Networks Using Learning Automata", in Proceeding of 3th Annual Int. Computer Society of Iran Computer Conf. CSICC-98, Tehran, Iran, pp. 417-428, 1999.

[64] A. A. Hashim, S. Amir and P. Mars, "Application of Learning Automata to Data Compression, In Adaptive and Learning Systems", K. S. Narendra (Ed), New York: Plenum Press, pp. 229-234, 1986.

[65] D. E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Reading, MA: Addition-Wesley, 1989.

[66] O. Sinnen, "Task Sheduling for Parallel Systems", Wiley, New Zealand, 2007.

[67] V. Sarkar, "Partitionning and Scheduling parallel Programs for Execution on Multiprocessors", MIT Press, Cambridge MA, 1989.

[68] M. R. Garey and D. S. Johnson, "Computers and Instractability: A Guid to Theory of P\Completeness", Freeman, New York,USA, 1979.

[69] M. I. Daoud and N. Kharma, "Ahigh performance algorithm for static task scheduling inheterogeneous distributed computing systems", J. Parallel Distrib. Comput. 68(2008) 399 – 409.

[70] M. Wu and D. Dajski, "Hypertool: a programming aid for message passing systems", IEEE Trans. Parallel Distributed Systems, vol 1, no. 3, 1990.

[71] R. Hwang, M. Gen and H. Katayama, "A comparison of multiprocessor task scheduling algorithms with communication costs", Computers & Operations Research 35 (2008)976 – 993.

[72] O. Sinnen, "Experimental Evaluation of Task Scheduling Accuracy", wiley , New Zealand, 2006.

[73] Y. K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors". ACM Comput. Surv. 31(4):406–471, 1999.

[74] H. Moti. G, Saeed Parsa and Mohammad Hossein Nejad, "Application of Learning Automata for DAG Scheduling on Homogeneous Networks", 17th Iranian Conference on Electrical Engineering, Iran University of Science and Technology,Tehran, Iran, ICEE2009.

[75] H. Moti. G, D.Keykhosravi and A. Hosseinalipour, "DAG Scheduling on Heterogeneous Distributed Systems using Learning Automata", Springer LNCS, Dong Hoi,Vietnam, 2010.

[76] H. Motee. G, K. Fakhr and S. Ahmadi "A New Method for task Scheduling", 2nd ICETC 2010, IEEE, Shanghai, China, 2010.

[77] H. Motee. G, A. Mirzaei and A. A. Dadjooyan,"Graph Coloring using Learning Automata", in Proceeding of 15th Annual Int. Computer Society of Iran Computer Conf.CSICC-2010, Tehran, Iran, 2010.

[78] H. Motee. G, K. Fakhr, M. Javadi, G. Bakhshzadeh, "Static Task Graph Scheduling Using Learner Genetic Algorithm", SoCPaR2010, IEEE, Paris, France, 2010.

واژه نامه فارسی به انگلیسی

الف	
آگاهی	Cognition
الگوریتم ژنتیک	Genetic Algorithm
الگوی بهینه سراسری	Global Best Model
ارث	Inheritance
ارزیابی بازگشتی سیاست	Iterative policy evaluation
ارزیابی سیاست	Policy evaluation
الگوی بهینه‌ی محلی	Local Best Model
اولویت	Priority
احتمالی	Probabilistic
انتخاب	Selection
ایستا	Static
احتمالی	Stochastic
آستانه	Threshold
اغتشاش	Jamming

ب	
برازندگی	Fitness
برچسب زنی نمونه‌ها	Relaxation labeling
برخورد	Conflict
بردار	Vector
برنامه‌نویسی پویا	Dynamic programming
بهبود سیاست	Policy improvement
بهینه	Optimal
بهینه‌سازی	Optimization
بهینه‌سازی انبوه ذرات	Partial Swarm Optimization
بیشینه	Maximum

پ	
پاداش	Reward
پاداش انتظاری	Expected reward
پاداش عددی	Numerical reward
پایداری	Stability
پردازش تصویر	Image processing
پردازنده	Processor
پس‌خورد	Feedback
پویا	Dynamic
پیچیدگی	Complexity
پیوستگی	Continuity

ت ث	
تابع	Function
تابع ارزش	Value Function
تابع ارزش بهینه	Optimal value function
تابع سیگموید	Sigmoid Function
تاریخچه	History
تبخیر فرمون	Pheromone evaporation
تپه‌نوردی	Hill climbing
تخصیص	Allocate
ترتیب وظایف	Task Sequence
ترجمه	Compile
ترکیب	Crossover
تصادف	Random
تصادفی	Probabilistic
تطبیقی	Adaptive
تعادل گرمایی	Thermal equilibrium
تعامل	Interact
تکامل	Evolution
تکامل سیاست	Policy iteration
ثابت	Fixed

ج	
جایگشت	Permutation
جریمه	Penalize
جستجو	Exploration
جستجوی سراسری	Global Search
جستجوی محلی	Local Search
جمعیت	Population
جهش	Mutation

چ	
چرخ رولت	Roulette Wheel
چرخشی	Circulate
چند جمله‌ای	Polynomial
چند عاملی	Multi Agent
چند پردازنده‌ای	Multi Processor

ح خ	
حالات پایانی	Terminating state
حالت	State
حالت بعدی	Successor states
حالت پایانی	Terminal state
حفره	Hole
خاصیت مارکوف	The markov property
خطی	Linear
خود-ترتیبی	Self-organization
خودکار	Automata
خودکار مهاجرت اشیا	Object Migration Automata

د ذ	
درخت	Tree
درک آنی	Immediate sensation
دودویی	Binary
دوره‌ای	Periodic
ذره	Bit

ر	
رابطه	Relation
راس	Vertex
رتبه‌بندی	Ranking
رشته	String
رکود	Stagnation
رنگ آمیزی گراف	Graph Coloring
روبات	Robot

ز ژ	
زمان اجرا	Run Time
زمان خاتمه	Finish Time
زمان محاسباتی	Computation time
زمان‌بندی	Scheduling
زمان‌بندی کار	Job Scheduling
زنجیر مارکوف	Markov chain
ژن	Gene

س	
سراسری	Global
سطح بالا	top-level
سطح پایین	Bottom-level
سوئیچ بسته‌ای	Packet switching
سوئیچ مداری	Circuit switching
سیاست	Policy
سیاست بهینه	Optimal policy
سیاست یا استراتژی عامل	Agents policy/strategy
سامانه مورچه	Ant System
سامانه‌های مرتبط	Connectionist systems

ش	
شانس	Chance
شبکه	Network
شبیه‌ساز	Simulation
شبیه‌ساز سرد کردن فلزات	Simulated Annealing
شتاب اجتماعی	Social acceleration
شطرنج	Chess
شیار	Valley

ط ع	
طبیعی	Nature
طول مسیر محاسباتی	Computation path length

عامل	Agent
عمق حافظه	Memory Depth
عمل	Action

ف	
فرآیند تصمیم‌گیری مارکوف	Markov decision process
فرآیند	Process
فرزند	Offspring
فرمون	Pheromone
فروشنده دوره گرد	Traveling Salesman
فضای جستجو	Search Space

ق	
قابل قبول بودن	Feasible
قانونی بودن	Legality
قطعی	Deterministic

ک	
کاوش	Exploitation
کدگذاری	Coding
کدگشایی	Decoding
کروموزوم	Chromosome
کمینه	Minimum

گ	
گراف بدون دور جهت-دار	Directed Acyclic Graph
گراف کامل	Complete Graph
گراف وظایف	Task Graph
گره خروجی	Exit node
گره ورودی	Entry node
گسستگی	Rupture

م	
مبدا	Source
متغیر	Variable
مجموعه	Set
محاسبات تکاملی	Evolution computation
محلی	Local
محیط	Environment
مرمت	Repair
مساله تخمین	Prediction problem
مستقل	Independent
مسیر	Path
مسیر بحرانی	Critical Path
مسیر بحرانی محاسباتی	Computation Critical Path
مسیریابی	Routing
معادله	Equation
معادله بهینگی	Optimality equation
معتبر	Valid
مقدار نگه‌دارنده	Clamping value
مقصد	Destination
مکاشفه	Heuristic
مورچه	Ant
مونت کارلو	MontCarlo
میزان اینرسی	inertial weight
میزان ذرات پراکنده شده	swarm size

ن	
نخبه‌کشی	Elitism
نرخ	Rate
نرخ تنزیل	Discount factor
نسل	Generation
نظریه بازی	Game theory
نقاب	Mask
نمودار گانت	Gantt Chart

ه ی	
هم‌گرایی	Convergence
هوش	Intelligence
یادگیری تقویتی	Reinforcement Learning
یال	Edge

واژه نامه انگلیسی به فارسی

A

Action	عمل
Adaptive	تطبیقی
Agent	عامل
Agents policy/strategy	سیاست یا استراتژی عامل
Allocate	تخصیص
Ant	مورچه
Ant System	سامانه مورچه
Automata	خودکار

B

Binary	دودویی
Bit	ذره
Bottom-level	سطح پایین

C

Chance	شانس
Chess	شطرنج
Chromosome	کروموزوم
Circuit switching	سوئیچ مداری
Circulate	چرخشی
Clamping value	مقدار نگه‌دارنده
Coding	کدگذاری
Cognition	آگاهی
Compile	ترجمه
Complete Graph	گراف کامل
Complexity	پیچیدگی
Computation Critical Path	مسیر بحرانی محاسباتی
Computation path length	طول مسیر محاسباتی
Computation time	زمان محاسباتی
Conflict	برخورد
Connectionist systems	سامانه‌های مرتبط
Continuity	پیوستگی
Convergence	هم‌گرایی
Critical Path	مسیر بحرانی
Crossover	ترکیب

D

Decoding	کدگشایی
Destination	مقصد
Deterministic	قطعی
Directed Acyclic Graph	گراف بدون دور جهت‌دار
Discount factor	نرخ تنزیل
Dynamic	پویا
Dynamic programming	برنامه‌نویسی پویا

E

Edge	یال
Elitism	نخبه‌کشی
Entry node	گره ورودی
Environment	محیط
Equation	معادله
Evolution	تکامل
Evolution computation	محاسبات تکاملی
Exit node	گره خروجی
Expected reward	پاداش انتظاری
Exploitation	کاوش
Exploration	جستجو

F

Feasible	قابل قبول بودن
Feedback	پس‌خورد

Finish Time	زمان خاتمه
Fitness	برازندگی
Fixed	ثابت
Function	تابع

G

Game theory	نظریه بازی
Gantt Chart	نمودار گانت
Gene	ژن
Generation	نسل
Genetic Algorithm	الگوریتم ژنتیک
Global	سراسری
Global Best Model	الگوی بهینه‌ی سراسری
Global Search	جستجوی سراسری
Graph Coloring	رنگ آمیزی گراف

H

Heuristic	مکاشفه
Hill climbing	تپه‌نوردی
History	تاریخچه
Hole	حفره

I

Image processing	پردازش تصویر
Immediate sensation	درک آنی
Independent	مستقل
Inertial weight	میزان اینرسی
Inheritance	ارث
Intelligence	هوش
Interact	تعامل
Iterative policy evaluation	ارزیابی بازگشتی سیاست

J

Jamming	اغتشاش
Job Scheduling	زمان‌بندی کار

L

Legality	قانونی بودن
Linear	خطی
Local	محلی
Local Best Model	الگوی بهینه‌ی محلی
Local Search	جستجوی محلی

M

Markov chain	زنجیر مارکوف
Markov decision process	فرآیند تصمیم‌گیری مارکوف
Mask	نقاب
Maximum	بیشینه
Memory Depth	عمق حافظه
Minimum	کمینه
MontCarlo	مونت کارلو
Multi Agent	چند عاملی
Multi Processor	چند پردازنده‌ای
Mutation	جهش

N

Nature	طبیعی
Network	شبکه
Numerical reward	پاداش عددی

O

Object Migration Automata	خودکار مهاجرت اشیا
Offspring	فرزند
Optimal	بهینه
Optimal policy	سیاست بهینه
Optimal value function	تابع ارزش بهینه
Optimality equation	معادله بهینگی
Optimization	بهینه‌سازی

P

Packet switching	سوئیچ بسته‌ای
Partial Swarm Optimization	بهینه‌سازی انبوه ذرات
Path	مسیر
Penalize	جریمه
Periodic	دوره‌ای
Permutation	جایگشت
Pheromone	فرمون
Pheromone evaporation	تبخیر فرمون
Policy evaluation	ارزیابی سیاست
Policy improvement	بهبود سیاست

Policy iteration	تکامل سیاست
Policy	سیاست
Polynomial	چند جمله‌ای
Population	جمعیت
Prediction problem	مساله تخمین
Priority	اولویت
Probabilistic	احتمالی
Process	فرآیند
Processor	پردازنده

R

Random	تصادف
Ranking	رتبه‌بندی
Rate	نرخ
Reinforcement Learning	یادگیری تقویتی
Relation	رابطه
Relaxation labeling	برچسب زنی نمونه‌ها
Repair	مرمت
Reward	پاداش
Robot	روبات
Roulette Wheel	چرخ رولت
Routing	مسیریابی
Run Time	زمان اجرا
Rupture	گسستگی

S

Scheduling	زمان‌بندی
Search	جستجو
Search Space	فضای جستجو
Selection	انتخاب
Self-organization	خود-ترتیبی
Set	مجموعه
Sigmoid Function	تابع سیگموید
Simulated Annealing	شبیه‌ساز سرد کردن فلزات
Simulation	شبیه‌ساز
Social acceleration	شتاب اجتماعی
Source	مبدا
Stability	پایداری
Stagnation	رکود
State	حالت
Static	ایستا
Stochastic	احتمالی
String	رشته
Successor states	حالت بعدی
Swarm size	میزان ذرات پراکنده شده

T

Task Graph	گراف وظایف
Task Sequence	ترتیب وظایف
Terminal state	حالت پایانی
The markov property	خاصیت مارکوف
Thermal equilibrium	تعادل گرمایی
Threshold	آستانه
top-level	سطح بالا
Traveling Salesman	فروشنده دوره گرد
Tree	درخت

V

Valid	معتبر
Valley	شیار
Value Function	تابع ارزش
Variable	متغیر
Vector	بردار
Vertex	راس

Islamic Azad University
Shabestar Branch

An Overview of some Intelligent Optimization Methods

در کتاب حاضر برخی از مهم ترین روش های بهینه سازی هوشمند به همراه کاربرد آن در حل مسایل مختلف معرفی شده است. مخاطبین اصلی این کتاب کسانی هستند که در یکی از رشته های تحصیلی علوم کامپیوتر، ریاضیات کاربردی، الکترونیک، الکتروتکنیک، کنترل، صنایع و مکاترونیک فعالیت دارند. البته این کتاب می تواند در رشته های مرتبط دیگری که به نحوی با مسایل بهینه سازی روبرو هستند نیز مورد استفاده قرار گیرد.

Compiled By

H. Motee. G

Sh. Lotfi

M. M. S. Esfahlan

شابک :978-964-10-0371-7

قیمت : ۶۰۰۰۰ ریال