فصل نهم: الگوریتم‌های ژنتیک

الگوریتم‌های ژنتیک روشی برای یادگیری‌هایی ارائه می‌دهد که از تکامل الهام گرفته شده است. بعضی مواقع فرضیه‌ها با رشته بیت‌هایی نمایش داده می‌شوند که تفسیر این رشته بیت‌ها به مسئله وابسته است، با این وجود فرضیه‌ها ممکن است حتی به صورت نشانه‌های نمادین[۱] یا برنامه‌های کامپیوتری بیان شوند. جستجو در میان فرضیه‌ها با یک جمعیت، مجموعه و یا فرضیه‌های اولیه آغاز می‌شود. اعضای جمعیت فعلی توسط اعمال تولیدمثل و جهشی که تصادفی انجام می‌شوند تغییر می‌یابند. این اعمال از اعمال مشابهی در تکامل زیستی الگوبرداری شده‌اند. در هر مرحله فرضیه‌های جمعیت فعلی با معیاری به نام تناسب ارزیابی می‌شوند. توابعی که تناسب بیشتری داشته باشند متناسباً احتمال بیشتری برای انتخاب برای تولیدمثل، جهش و یا حضور مستقیم در جمعیت نسل بعد خواهند داشت. الگوریتم‌های ژنتیک در مسائل یادگیری و غیر یادگیری زیادی به کار رفته‌اند. برای مثال، از آن‌ها برای یادگیری دسته قوانین برای کنترل ربات و بهینه‌سازی توپولوژی و یادگیری پارامترهای شبکه‌های عصبی استفاده می‌شود. در این فصل هم به الگوریتم‌های ژنتیک، که از فرضیه‌های رشته بیتی استفاده می‌کنند، و هم به برنامه‌نویسی ژنتیک، که فرضیه‌هایش برنامه‌های کامپیوتری‌اند، می‌پردازیم.

۹,۱ انگیزه

اساس انگیزه‌ی الگوریتم‌های ژنتیک[۲] یا GA ها تکامل زیستی است. به جای استفاده از ترتیب کلی‌تری یا ترتیب ساده به پیچیده، الگوریتم‌های ژنتیک با توسعه دادن و ترکیب فرضیه‌های درست‌تر شناخته شده به فرضیه‌های درست جدیدتری می‌رسند. در هر مرحله مجموعه‌ای از فرضیه‌ها ، جمعیت[۳]، در نظر گرفته می‌شود. در هر مرحله کسری از جمعیت با فرزندانی که از بهترین فرضیه‌ها توسط عمل تولیدمثل[۴] ایجاد می‌شود جایگزین می‌شوند. چنین فرایندی یک سری آزمون‌وخطا را ایجاد می‌کند، در هر مرحله قدرت فرضیه‌ی ایجاد شده مورد آزمون قرار

---

[۱] symbolic expressions
[۲] Genetic algorithms
[۳] population
[۴] offspring

می‌گیرد. در تولید فرضیه‌های جدید معمولاً از خواصی از فرضیه‌های بهتر استفاده می‌شود. این نوع بررسی در الگوریتم‌های ژنتیک از چندین عامل انگیزه گرفته است:

- تکامل در طبیعت به عنوان یکی از متدهای موفق تطبیق‌پذیری پذیرفته شده است.
- الگوریتم ژنتیک می‌توانند فضاهای فرضیه‌ای را که ویژگی‌های متقابل پیچیده‌ای دارند جستجو کند. در چنین فرضیه‌هایی تغییر هر یک از خواص فرضیه تأثیر بسزایی در کل سازگاری فرضیه دارد.
- این الگوریتم‌ها هزینه‌ی استفاده از ابرکامپیوترها را ندارند، الگوریتم‌های ژنتیک را می‌توان به سادگی به چندین قسمت تقسیم و با کامپیوترهای مجزایی بررسی کرد.

در این بخش الگوریتم‌های ژنتیک را توضیح داده، کاربردهایشان و طبیعت جستجوی فضای فرضیه‌ای‌شان را بررسی می‌کنیم. همچنین به مقوله‌ی برنامه‌نویسی ژنتیک نیز می‌پردازیم، مقوله‌ای که در آن تمامی برنامه‌های کامپیوتری تا اندازه‌ی خاصی تکامل پیدا می‌کنند. الگوریتم‌های ژنتیک و برنامه‌نویسی ژنتیک هر دو زیرمجموعه‌ی محاسبات تکاملی[۱] هستند. در قسمت آخر نیز سر تیترها در مطالعه‌ی تکامل زیستی، از جمله اثر بالدوین را بررسی می‌کنیم و رابطه‌ی بین تکامل افراد و تکامل کل جمعیت را نیز بررسی خواهیم کرد.

۹,۲ الگوریتم‌های ژنتیک

در این مسئله که توسط الگوریتم‌های ژنتیک مطرح می‌شود، مرحله‌ی اول پیدا کردن فضایی از فرضیه‌های کاندید است تا در بین آن‌ها فرضیه‌هایی با بهترین عملکرد را پیدا کنیم. در الگوریتم‌های ژنتیک تعریف "بهترین فرضیه" از این قرار است: فرضیه‌ای که معیارهای پیش تعریف شده عددی‌ای را که تابع تناسب[۲] نامیده می‌شود را برای مسئله‌ی موجود بهینه کند (حداکثر یا حداقل). برای مثال، اگر کار یادگیری تخمین یک تابع مجهول با نمونه‌های آموزشی ورودی و خروجی آن است، تابع تناسب می‌تواند به صورت دقت فرضیه بر روی نمونه‌های آموزشی تعریف شود. یا اگر هدف یادگیری روشی برای شطرنج بازی کردن است تابع تناسب می‌توان تعداد برد هر فرد در مقابل افراد دیگر در همان جمعیت باشد.

با اینکه الگوریتم‌های ژنتیک در کاربردهای متفاوتی به کار می‌روند و در جزئیات با هم متفاوت‌اند اما معمولاً در ساختارهای ذیل مشابه‌اند: در هر مرحله الگوریتم مجموعه‌ای از فرضیه‌ها را به نام جمعیت تغییر می‌کند، در هر مرحله تمامی فرضیه‌های جمعیت با تابع تناسب مورد بررسی قرار می‌گیرند، جمعیت جدید با انتخاب تصادفی چند فرضیه از فرضیه‌های بهتر جمعیت ایجاد می‌شود، تعدادی از این فرضیه‌ها مستقیماً به نسل بعد منتقل می‌شوند و بقیه در تولیدمثل فرضیه‌های جدید از طریق اعمال ژنتیکی‌ای چون تولیدمثل[۳] و جهش[۴] مورد استفاده قرار می‌گیرند.

حالت کلی الگوریتم‌های ژنتیک در جدول ۹,۱ آورده شده است. ورودی‌های این الگوریتم شامل تابع تناسب، برای رده‌بندی فرضیه‌ها، مقدار آستانه، برای تشخیص میزان تناسب و پایان دادن به الگوریتم، تعداد جمعیتی که باید باقی بمانند و پارامترهایی مربوط به تشکیل جمعیت‌های موفق است: درصدی از جمعیت که جایگزین می‌شوند و سرعت جهش.

---

۱ evolutionary computation
۲ fitness function
۳ crossover
۴ mutation

GA(Fitness,Fitness_treshold,p,r,m)

Fitness: تابعی که به فرضیه‌ها مقداری برای ارزیابی نسبت می‌دهد.

Fitness_treshold: مقدار آستانه‌ای که شرط پایانی را مشخص می‌کند

p: تعداد فرضیه‌ای که باید در هر جمعیت باشد

r: نسبتی از جمعیت که در هر مرحله با استفاده از تولیدمثل جایگزین می‌شوند

m: ضریب جهش

- مقداردهی اولیه: جمعیت P را با فرضیه‌های اتفاقی‌ای ایجاد کن.
- ارزیابی: برای هر فرضیه‌ی h در P مقدار Fitness(h) را محاسبه کن.
- تا زمانی که $\max_h Fitness(h) < Fitness\_threshold$ حلقه ی زیر را اجرا کن.

  نسل جدید P_s را ایجاد کن:

  ۱. انتخاب: با توزیع احتمال زیر (1-r)p عضو از P را به P_s اضافه کن.

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{h=1}^{p} Fitness(h_j)}$$

  ۲. تولیدمثل: با توزیع احتمال بالا را به $\frac{r.p}{2}$ جفت فرضیه انتخاب کن. برای هر جفت $< h_1, h_2 >$ با استفاده از عمل تولیدمثل دو فرزند ایجاد کن و در مجموعه‌ی P_s قرار بده.

  ۳. جهش: m عضو از P_s را با توزیع احتمال یکنواخت انتخاب کن و یکی از بیت‌های نمایشش را به دلخواه عوض کن.

  ۴. تغییر: $P \leftarrow P_s$

  ۵. ارزیابی: برای هر فرضیه‌ی h در P مقدار Fitness(h) را محاسبه کن.

- فرضیه‌ای در P که بالاترین تناسب را دارد را خروجی بده.

جدول ۹,۱ حالت کلی الگوریتم‌های ژنتیک.

در انتها جمعیتی با p فرضیه باقی می‌ماند. در هر بار تکرار حلقه جمعیت موفق‌تر $\boldsymbol{P_s}$ با انتخاب احتمالی فرضیه‌ها و اضافه کردن فرضیه‌های جدید شکل می‌گیرد. فرضیه‌های جدید از عمل تولیدمثل بر روی جفت فرضیه‌ها ایجاد می‌شوند و بر روی بعضی از فرضیه‌ها نیز عمل جهش اتفاق می‌افتد. این فرایند تا زمانی که فرضیه با متناسب مورد نظر ایجاد شود ادامه دارد. اعمال تولیدمثل و جهش معمول در جدولی در ادامه‌ی بحث آورده شده‌اند.

توجه داشته باشید که در این الگوریتم هر بار اجرای حلقه‌ی اصلی نسلی جدید از فرضیه‌ها را از روی جمعیت فعلی ایجاد می‌کند. در گام اول، تعداد خاصی از فرضیه‌ها از جمعیت فعلی انتخاب می‌شوند تا نسل بعدی را تشکیل دهند. چنین فرضیه‌هایی با توزیع احتمال زیر انتخاب می‌شوند:

$$\Pr(h_i) = \frac{Fitness(h_i)}{\sum_{h=1}^{p} Fitness(h_j)} \qquad (9.1)$$

پس احتمال انتخاب هر فرضیه با تناسبش رابطه‌ی مستقیم و با مجموع تناسب فرضیه‌های رقیب رابطه‌ی عکس دارد.

زمانی که اعضای نسل فعلی برای تشکیل نسل بعد انتخاب می‌شوند، از طریق عمل تولیدمثل فرضیه‌های جدیدی نیز ساخته و اضافه می‌شوند. عمل تولیدمثل، انتخاب دو فرضیه به عنوان والدین و ترکیب قسمت‌های مختلف آن‌هاست، در قسمت‌های آینده عمل تولیدمثل را مفصلاً توضیح خواهیم داد. فرضیه‌های والد به صورت تصادفی از جمعیت فعلی با توزیع احتمال رابطه‌ی ۹,۱ انتخاب می‌شوند. بعد از تولیدمثل و ایجاد اعضای جدید، نسل جدید جمعیت تعداد کافی اعضا را خواهد داشت. سپس کسر خاصی از این اعضا به صورت اتفاقی انتخاب می‌شوند و عمل جهش بر روی آن‌ها انجام می‌گیرد.

این الگوریتم ژنتیک جستجویی موازی و ستونی[۱] در میان فرضیه‌هایی که تناسب بهتری دارند انجام می‌دهد. در قسمت‌های بعدی، فرضیه‌ها و اعمال ژنتیک را دقیق‌تر بررسی خواهیم کرد.

۹,۲,۱ معرفی فرضیه‌ها

فرضیه‌هایی که در الگوریتم‌های ژنتیک استفاده می‌شوند معمولاً به صورت رشته‌های بیت نمایش داده می‌شوند تا بتوان اعمال تولیدمثل و جهش را به راحتی روی آن‌ها انجام داد. این نمایش فرضیه‌ها می‌تواند بسیار پیچیده باشد. برای مثال، مجموعه‌ی دستورهای if-then را می‌توان با در نظر گرفتن کدی برای هر دستور به راحتی در این نمایش نشان داد. مثال‌های انواع نمایش این نوع دستورها در (Holland 1986)، (Grefenstette 1988) و (DeJong 1993) و دیگر مقالات آورده شده است.

برای تصور اینکه چگونه می‌توان دستورهای if-then را با رشته بیت‌ها نمایش داد، ابتدا طرز نمایش یک شرط[۲] را بر روی مقدار یک ویژگی در نظر بگیرید. برای مثال، ویژگی outlook را که سه مقدار Sunny، Rainy و Cloudy را می‌تواند بپذیرد در نظر بگیرید. یکی از ساده‌ترین راه‌های نمایش چنین ویژگی‌ای قرار دادن ۱ در مکان مربوطه‌ی هر یک از این مقادیر است. مثلاً رشته‌ی ۰۱۰ نشان می‌دهد که outlook=Cloudy است. به طور مشابه اگر رشته ۰۱۱ باشد نشان‌دهنده‌ی این است که outlook یکی از دو مقدار ممکن را دارد (outlook=Rainy v Cloudy). توجه داشته باشید که کلی‌ترین فرضیه در چنین نمایشی 111 است که نشان‌دهنده‌ی این است که مقدار ویژگی برای فرضیه تفاوتی نمی‌کند.

با توجه به این متد برای نمایش ویژگی‌ها، می‌توان روابط فصلی را با به هم پیوستن رشته‌ها نشان داد. برای مثال ویژگی دوم Wind را با دو مقدار Strong و Weak در نظر بگیرید. فرضیه‌ای مثل فرضیه‌ی

$$(\text{outlook} = \text{Cloudy} \vee \text{Rainy}) \wedge (\text{Wind} = \text{Strong})$$

را می‌توان به راحتی به شکل پنج بیت زیر نشان داد:

Outlook	Wind
011	10

حکمی (PlayTennis = yes) را می‌توان به همین صورت نشان داد. پس کل رابطه را می‌توان با سرهم کردن بیت‌های شرط و حکم نشان داد. مثلاً رابطه‌ی

---

[۱] beam search

[۲] constraint

IF Wind = Strong THEN PlayTennis = yes

به صورت زیر نمایش داده خواهد شد:

Outlook	Wind	PlayTennis
011	10	10

که در آن سه بیت اول نشان‌دهنده‌ی عدم اهمیت Outlook و دو بیت بعدی وضعیت Wind و دو بیت آخر حکم را نشان می‌دهد (در اینجا فرض کرده‌ایم که PlayTennis دو مقدار yes و no را می‌تواند داشته باشد). توجه داشته باشید که نمایش بیتی قوانین برای هر ویژگی در فرضیه یک زیررشته[۳] در نظر می‌گیرد حتی اگر آن ویژگی جزو ویژگی‌های شرط نباشد. همین امر باعث می‌شود که همیشه فرضیه‌ها، رشته‌های بیتی‌هایی با طول ثابت باشند که جای هر یک از زیررشته‌ها نیز در آن ثابت است. با چنین نمایشی برای قوانین، می‌توان دسته قوانین[۴] را با پشت سر هم آوردن چنین نمایشی از قوانین ساخت.

بد نیست که در طراحی رشته بیت‌ها برای کد سازی فضای فرضیه‌ای برای هر مقدار مجاز هر ویژگی یک بیت در نظر بگیریم تا فرضیه‌ها قابلیت ارتجاع داشته باشند. برای درک بهتر، توجه دارید که در بالا در یکی از قوانین ذکر شده (11 10 111) هیچ اطلاعاتی درباره‌ی ویژگی PlayTennis به ما نمی‌دهد. اگر بخواهیم از چنین نمونه‌هایی دوری کنیم می‌توانیم نمایش دیگری را انتخاب کنیم (مثلاً فقط یک بیت را به بازی تنیس اختصاص داده و برای مقدار بله ۱ و برای مقدار خیر ۰ را در نظر بگیریم)، اما بهتر است اعمال ژنتیکی را طوری تغییر دهیم که چنین فرضیه‌هایی ایجاد نشوند یا به روش دیگر تناسب پایینی را به چنین فرضیه‌هایی نسبت دهیم.

در بعضی الگوریتم‌های ژنتیک، فرضیه‌ها با نشانه‌های نمادین (به جای رشته بیت‌ها) نمایش داده می‌شوند. برای مثال، در قسمت ۹٫۵ الگوریتم ژنتیکی را بررسی خواهیم کرد که فرضیه‌هایش برنامه‌های کامپیوتری هستند.

۹٫۲٫۲ اعمال ژنتیکی

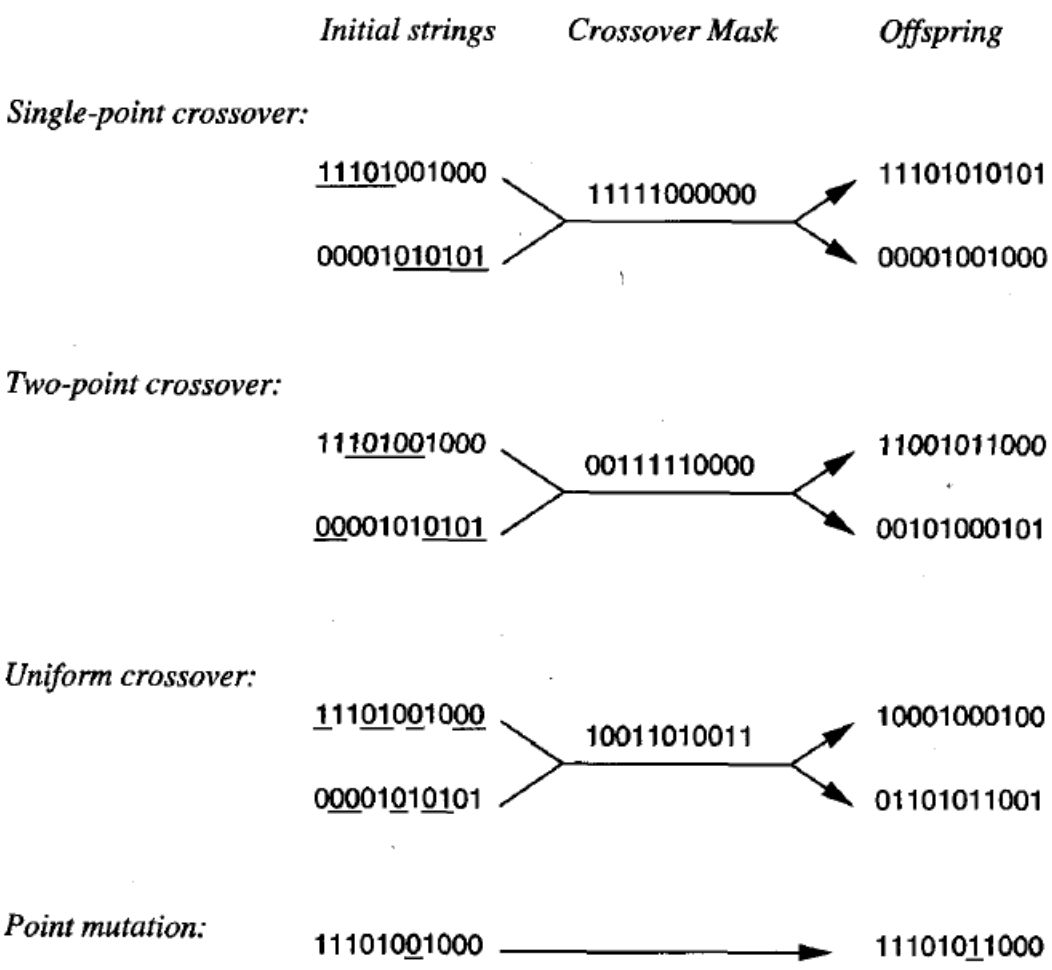
ایجاد جمعیت‌های جدید در الگوریتم‌های ژنتیک با استفاده از اعمالی چون تولیدمثل و جهش انجام می‌شود. اعمال متداول در الگوریتم‌های ژنتیک برای ایجاد رشته بیت‌های جدید در جدول ۹٫۱ آورده شده است. این عملگرها متناسب با اعمال ژنتیکی ایده آل سیستم‌های تکامل زیستی طراحی شده‌اند. معمول‌ترین این اعمال تولیدمثل و جهش است.

عمل تولیدمثل دو فرزند را از دو والد با ترکیب کپی بیت‌هایشان ایجاد می‌کند. هر بیت در موقعیت i ام فرزندان کپی بیتی در همان موقعیت یکی از والدینشان است. انتخاب اینکه کدام بیت را از کدام والد انتخاب می‌کنیم توسط رشته‌ای دیگر به نام نقاب تولیدمثل[۵] انجام می‌شود. برای تصور، تولیدمثل تک نقطه‌ای[۶] که در بالای جدول ۹٫۲ آمده است را در نظر بگیرید. به فرزند بالایی توجه کنید، این فرزند تمامی پنج بیت اول را از والد اول و شش بیت باقی‌مانده را از والد دوم به ارث برده است، زیرا که نقاب تولیدمثلش ۱۱۱۱۱۰۰۰۰۰۰ بوده است. فرزند دوم نیز از عکس همان نقاب استفاده کرده است، به همین دلیل فرزند دوم بیت‌هایی را کپی کرده که فرزند اول از آن‌ها استفاده‌ای نکرده بود. همیشه در تولید مثال تک نقطه‌ای ابتدای نقاب n بیت 1 و بقیه ۰ هستند. به همین دلیل فرزندان n بیت اول از یکی از والدین و بقیه از والد دیگر به ارث ببرند.

---

[۳] substring
[۴] sets of rules
[۵] crossover mask
[۶] single-point crossover

هر بار که تولیدمثل تک نقطه‌ای به کار برده می‌شود، ابتدا نقطه‌ی تولیدمثل n به طور تصادفی انتخاب می‌شود سپس نقـاب سـاخته شـده و در تولیدمثل به کار برده می‌شود.



جدول ۹,۲ اعمال معمول در الگوریتم‌های ژنتیک.

این اعمال فرزندانی را با استفاده از نمایش بیتی فرضیه‌ها ایجاد می‌کنند. عمل تولیدمثل دو فرزند را از دو والد تولید می‌کند، نقاب معلوم می‌کند که هر فرزند کدام بیت‌ها را به ارث ببرد. جهش با تغییر یکی از بیت‌ها از یک والد یک فرزند ایجاد می‌کند.

در تولیدمثل دونقطه‌ای[۷] فرزندان قسمتی از وسط بیت‌ها را از یکی از والدین و بقیه را از والدی دیگر دریافت می‌کننـد. بـه عبـارت دیگـر نقـاب تولیدمثل دونقطه‌ای رشته بیتی است که با n_0 بیت ۰ شروع شده و با n_1 بیت ۱ ادامه پیدا می‌کند و بقیه بیت‌ها نیز ۰ خواهد بود. هـر بـار کـه تولیدمثل دونقطه‌ای به کار می‌رود ابتدا دو عدد n_0 و n_1 به صورت تصادفی انتخاب شده سپس نقاب لازم ایجاد و تولیدمثل انجام می‌شود. مثلاً در مثالی که در جدول ۹,۲ آمده $n_0 = 2$ و $n_1 = 5$ است. مشابه دیگر تولیدمثل‌ها، فرزند دوم با جابجا کردن نقش دو والد ایجاد می‌شود.

[۷] two-point crossover

تولیدمثل یکنواخت[۸] به طور یکنواخت بیت‌های والدین را با هم ترکیب و فرزندان را ایجاد می‌کند (همان‌طور که در جدول ۹,۲ نیز نشان داده شده است). در چنین تولیدمثلی نقاب به صورت کاملاً تصادفی ایجاد می‌شود (هر بیت نقاب به طور تصادفی و مستقل از بقیه بیت‌ها انتخاب می‌شود).

علاوه بر اعمال تولیدمثل که از دو والد برای ایجاد فرزند استفاده می‌کنند، عمل دیگری که از یک والد فرزند ایجاد می‌کند نیز وجود دارد. در کل، جهش تغییر کوچکی در فرضیه‌هاست که با تغییر یک بیت به وجود می‌آید. در بعضی سیستم‌ها بجای الگوریتم جدول ۹,۱ از الگوریتم‌هایی استفاده می‌شود که بعد از تولیدمثل جهش را اعمال می‌کنند.

بعضی از سیستم‌های ژنتیکی اعمال دیگری را نیز به کار می‌گیرند که منحصر به طرز نمایش فرضیه‌هایشان است. برای مثال، (Grefenstette 1991) سیستمی را معرفی می‌کند که دسته قوانینی را برای کنترل ربات یاد می‌گیرد، این سیستم علاوه بر اعمال تولیدمثل و جهش، از عملی اضافی برای خاص کردن قوانین کمک می‌گیرد. (Janikow 1993) نیز سیستمی را معرفی می‌کند که دسته قوانینی را با استفاده از اعمال کلی سازی و خاص سازی قوانین از لحاظ‌های مختلف به کار می‌گیرد. (برای مثال با تغییر یکی از شروط رابطه‌ی if-then با عامل "فرقی نمی‌کند").

۹,۲,۳ تابع تناسب و انتخاب

تابع تناسب معیاری برای ترتیب کردن فرضیه‌ها و انتخاب تصادفی آن‌ها برای حضور در نسل بعدی جمعیت است. اگر هدف یادگیری دسته قوانین باشد، تابع تناسب قسمتی خواهد داشت تا دقت قانون را بر روی نمونه‌های آموزشی موجود اندازه‌گیری کند. بعضی موارد معیارهای دیگری نیز در تابع تناسب تأثیر گذارند (مثلاً پیچیدگی قانون یا کلی بودن قانون). در کل برای فرضیه‌های رشته بیتی‌ای که فرایند پیچیده‌ای دارند (مثلاً رشته بیتی که از تعداد زیادی قانون زنجیروار if-then تشکیل یافته تا یک دستگاه رباتیک را کنترل کند)، تابع تناسب کارایی کلی فرضیه را در نظر می‌گیرد.

در حالت کلی در الگوریتم ژنتیکی که در جدول ۹,۱ آورده شده، احتمال انتخاب یک فرضیه با تناسب خودش نسبت مستقیم و با تناسب دیگر فرضیه‌های جمعیت فعلی نسبت عکس دارد (رابطه‌ی ۹,۱). این متد را گاهی انتخاب نسبی تناسبی[۹] و یا رولت[۱۰] می‌نامند. متدهای دیگری نیز برای انتخاب فرضیه‌ها بر حسب تناسبشان ارائه شده است. برای مثال، در انتخاب مسابقه‌ای[۱۱] ابتدا دو فرضیه به صورت اتفاقی از جمعیت فعلی انتخاب می‌شوند. سپس با احتمالی از پیش تعریف شده مثل p فرضیه‌ی متناسب‌تر و با احتمال (1-p) فرضیه‌ای که تناسب کمتری دارد انتخاب می‌شود. با چنین انتخابی معمولاً به جای اینکه فرضیه‌های متناسب‌تر انتخاب شوند فرضیه‌های گوناگونی انتخاب می‌شوند (Goldberg and Deb 1991). در متد دیگری که انتخاب رتبه‌ای[۱۲] نامیده می‌شود، در ابتدا تمامی فرضیه‌های جمعیت فعلی متناسب با تناسبشان ترتیب شده و سپس متناسب با رتبه‌هایشان احتمالی برای انتخاب شدن می‌گیرند (و نه متناسب با تناسبشان).

---

[۸] uniform crossover
[۹] fitness proportionate selection
[۱۰] roulette wheel selection
[۱۱] tournament selection
[۱۲] rank selection

۹,۳ یک مثال

می‌توان به الگوریتم ژنتیک به دید یک متد بهینه برای جستجوی فضای بزرگی از اشیاء با هدف پیدا کردن متناسب‌ترین فرضیه‌ها (بر اساس تابع تناسب) نگاه کرد. با این وجود هیچ تضمینی نیست که خروجی این الگوریتم‌ها همیشه متناسب‌ترین فرضیه باشد، فقط می‌توان گفت که معمولاً خروجی این الگوریتم‌ها فرضیه‌هایی با تناسب بالاست. الگوریتم‌های ژنتیک در مسئله‌هایی در خارج قلمروی یادگیری ماشین مثل طراحی مدار[۱۳] و برنامه‌ریزی مغازه‌داری[۱۴] نیز به کار گرفته شده‌اند. در داخل قلمرو یادگیری ماشین نیز هم در تخمین توابع و هم در مسائلی همچون انتخاب نوع شبکه‌های عصبی به کار رفته‌اند.

برای تصور بهتر از کاربرد الگوریتم‌های ژنتیک در یادگیری مفهوم، در اینجا به طور خلاصه به سیستم GABIL که توسط (DeJong 1993) معرفی شده می‌پردازیم. در GABIL از الگوریتم ژنتیک برای یادگیری یک مفهوم منطقی با قانون‌هایی که قانون‌های فصلی گزاره‌ای[۱۵]، استفاده شده است. در آزمایش‌های انجام شده بر روی مسائل مختلف یادگیری مفهوم، GABIL قدرت تامیم قابل‌توجهی داشته است. این قدرت تامیم را بعداً با قدرت تامیم الگوریتم‌های C4.5 و AQ14 مقایسه می‌کنیم. در این تحقیق هم از مسائل مصنوعی و هم از نمونه‌های واقعی در تشخیص سرطان سینه برای مشاهده‌ی قدرت تامیم استفاده شده است.

الگوریتم به کار رفته در GABIL همان الگوریتم جدول ۹,۱ است. در تحقیقات (DeJong 1993)، پارامتر r، کسری از جمعیت را که به نسل بعدی منتقل می‌شوند 0.6 و پارامتر m، ضریب جهش 0.001 بوده است. (چنین شرایطی، شرایطی متداول محسوب می‌شوند). و تعداد اعضای جمعیت نیز بسته به مسئله بین ۱۰۰ تا ۱۰۰۰ بوده است.

اطلاعات خاص الگوریتم ژنتیک بکار رفته در GABIL مختصراً به شرح زیر است:

- **نمایش.** هر فرضیه در GABIL مجموعه‌ای از گزاره‌های فصلی که در قسمت ۹,۲,۱ مفصلاً به آن‌ها پرداختیم است. در کل، فضای فرضیه‌ای از قانون‌های گزاره‌ای‌ای که از روابط فصلی بر روی تعداد خاصی از ویژگی‌ها تشکیل شده است. برای نمایش یک دسته قانون رشته بیت‌های قانون‌های مختلف به پشت هم می‌آیند. برای تصور، فرض کنید که فضای فرضیه‌ای داریم که دو ویژگی منطقی به نام‌های a_1 و a_2 دارند و تابع هدف نیز یک تابع منطقی به نام c است پس دو قانون زیر

$$IF\ a_1 = T \wedge a_2 = F \text{ Then } c = T \text{ ; IF } a_2 = T \text{ Then } c = F$$

به فرم نشان داده شده نشان داده می‌شوند:

a_1	a_2	c	a_1	a_2	c
۱۰	۰۱	۱	۱۱	۱۰	۰

توجه دارید که طول رشته متناسب با تعداد قوانین بکار رفته در فرضیه تغییر می‌کند. با متغیر بودن طول نمایش رشته‌ی بیت لازم می‌شود که عمل تولیدمثل متناسب با آن تغییر کند:

---

[۱۳] circuit layout

[۱۴] job-shop scheduling

[۱۵] disjunctive set of propositional rules

- **اعمال ژنتیکی.** GABLIL از همان تعریف جهش که در جدول ۹,۲ آمده بود استفاده می‌کند، عمل تولیدمثل نیز یک تولیدمثل دونقطه‌ای ساده است که در جدول ۹,۲ نیز توضیح داده شده بود. در کل برای تطبیق با این حقیقت که طول رشته‌ها متغیر است و همچنین برای اینکه بیت‌ها در فرزندان نیز در جای خود باشند، از روش ذیل استفاده می‌شود: برای انجام چنین تولیدمثلی از دو والد ابتدا نقاط تولیدمثل به صورت تصادفی انتخاب می‌شوند. اگر d_1 و d_2 دو فاصله‌ی نقاط از چپ و راست رشته بیت‌ها (هر یک از قوانین نه کل فرضیه) باشد، مسئله‌ی مهم این است که این نقاط برای والد دوم باید d_1 و d_2 مشابهی داشته باشند، مثلاً اگر دو والد به شکل

	a_1	a_2	c	a_1	a_2	c
h_1:	۱۰	۰۱	۱	۱۱	۱۰	۰

و

	a_1	a_2	c	a_1	a_2	c
h_2:	۰۱	۱۱	۰	۱۰	۰۱	۰

باشند و اگر نقاط تولیدمثل برای والد اول ۱ و ۸ باشد:

	a_1	a_2	c	a_1	a_2	c
h_1:	۱[۰	۰۱	۱	۱۱	۱]۰	۰

برای این نقاط دو مقدار $d_1 = 1$ و $d_2 = 3$ هستند. بنابراین احتمال‌های موجود برای انتخاب دونقطه‌ی تولیدمثل والد دوم با توجه به مقادیر محدود به <1,3>، <1,8> و <6,8> می‌شود. حال اگر برای والد دوم نیز <1,3> انتخاب شود داریم که:

	a_1	a_2	c	a_1	a_2	c
h_2:	۰[۱	۱]۱	۰	۱۰	۰۱	۰

پس دو فرزند به شکل‌های

	a_1	a_2	c
h_3:	۱۱	۱۰	۰

و

	a_1	a_2	c	a_1	a_2	c	a_1	a_2	c
h_4:	۰۰	۰۱	۱	۱۱	۱۱	۰	۱۰	۰۱	۰

خواهند بود.

همان‌طور که در مثال نیز نشان داده شد، این روش باعث می‌شود تا فرزندان بتوانند تعداد متغیری (نه الزاماً مساوی والدینشان) قانون داشته باشند. این روش همچنین تضمین می‌کند که تمامی فرضیه‌های تولید شده معنی‌دار هستند.

- **تابع تناسب.** تناسب هر فرضیه را بر اساس دقت دسته‌بندی نمونه‌های آموزشی می‌سنجند. در اینجا تابع استفاده شده برای پیدا کردن تناسب هر فرضیه به شکل زیر است:

$$Fitness(h) = \big(correct(h)\big)^2$$

در این رابطه correct(h) تعداد نمونه‌های آموزشی‌ای است که h درست دسته‌بندی می‌کند.

در آزمایش مقایسه‌ی رفتار GABIL و الگوریتم‌های یادگیری درختی‌ای مثل C4.5 و ID5R و قانون آموزش AQ14 که (DeJong 1993) مطرح کرده تفاوت‌های قابل‌مقایسه‌ای در عملکرد این سیستم‌ها بر روی مسائل مختلف دیده می‌شود. برای مثال، برای ۱۲ مسئله‌ی ترکیبی، عملکرد الگوریتم GABILL ۹۲٫۱٪ بود در حالی که دیگر سیستم‌ها عملکردی بین ۹۱٫۲٪ تا ۹۶٫۶٪ داشتند.

۹٫٤ جستجوی فضای فرضیه‌ای

همان‌طور که در بالا نیز نشان داده شد، الگوریتم‌های ژنتیک با استفاده از یک متد جستجوی ستونی به دنبال فرضیه‌ای که تناسب حداکثر را داشته باشند می‌گردند. این جستجو با دیگر متدهای یادگیری که در این کتاب آمده متفاوت است. مثلاً در Backpropagation در شبکه‌های عصبی این جستجو با تغییر اندک‌اندک در فرضیه انجام می‌شد، در حالی که در الگوریتم ژنتیک این تغییرات به شدت و ناگهانی است، همان‌طور که واضح است فرزندان ممکن است تفاوت بسیاری با والدین داشته باشند. توجه داشته باشید که در الگوریتم‌های ژنتیک احتمال اینکه در مینیمم نسبی به دام بیفتیم بسیار کم است (مشکل اصلی Backpropagation).

یکی دیگر از تفاوت‌های کاربردی الگوریتم ژنتیک مشکل تراکم[۱۶] است. تراکم پدیده‌ای است که در آن افرادی که تناسب بالاتری دارند سریعاً تولیدمثل می‌کنند و تمامی جمعیت را با کپی‌های مشابه خود پر می‌کنند. مشکل در اینجاست که با کم شدن تنوع سرعت تکامل نیز کاهش می‌یابد. راه‌های بسیاری برای مقابله با مشکل تراکم ارائه شده است. یکی از این راه‌ها تغییر تابع انتخاب است، مثلاً می‌توان از انتخاب مسابقه‌ای یا انتخاب رتبه‌ای به جای رولت استفاده کرد. استراتژی مشابهی نیز به نام اشتراک تناسب[۱۷] وجود دارد که تناسب یک فرد را با افزایش افراد مشابه کم می‌کند. راه سوم محدود کردن اجازه‌ی افراد برای جفت‌گیری و تولیدمثل است، حتی می‌توانیم در بین نمونه‌های مشابه تعدادی را حذف کنیم یا زیرگونه‌های[۱۸] مشابه و متعدد را حذف کنیم. راه‌حل مشابه معلوم کردن فاصله‌ی فرضیه‌ها و اجازه‌ی جفت‌گیری به افراد نزدیک به هم است. بسیاری از این تکنیک‌ها از تکامل زیستی نشأت می‌گیرند.

۹٫٤٫۱ تکامل جمعیت و تئوری الگو[۱۹]

حال این سؤال مطرح است که آیا می‌توان ریاضی‌وار سیر تکامل جمعیت در الگوریتم ژنتیک در طول زمان را مشخص کرد. تئوری الگوی (Holland 1975) یک توصیف از تکامل جمعیت ارائه می‌کند. این توصیف مبتنی بر مفهوم الگو[۲۰] است که دسته رشته بیت‌ها را توصیف می‌کند. دقیق‌تر اینکه یک الگو رشته‌ای از ۰ و ۱ و * است. هر الگو دسته‌ای از رشته کدها را مشخص می‌کند که همان ۰ و ۱ ها را دارند و مقدار * نیز برایشان مهم نیست. برای مثال الگوی 10*0 دسته رشته‌ی شامل دو رشته بیت 0010 و 0110 را نشان می‌دهد.

هر رشته بیت را می‌توان نمایشگر تمامی الگوهایی که آن را توصیف می‌کند دانست. برای مثال، 0010 را می‌توان نمایشگر 2^4 الگو مثل **00، 10*0 و **** و غیره دانست. به طور مشابه جمعیتی از رشته کدها را می‌توان با تعدادی الگو و تعداد اعضای متناسب با آن الگو مشخص کرد.

---

[۱۶] Crowding
[۱۷] fitness sharing
[۱۸] subspecies
[۱۹] Scheme Theorem
[۲۰] Schema/pattern

تئوری الگو می‌تواند مشخصه پدیده تکامل جمعیت در الگوریتم ژنتیک را بر اساس تعداد نمونه‌های هر الگو بیان نماید. اگر m(s,t) تعداد نمونه‌های الگوی s در جمعیت در زمان t باشد (تکرار t ام الگوریتم)، تئوری الگو با توجه به m(s,t) و دیگر ویژگی‌های الگو و جمعیت و پارامترهای الگوریتم ژنتیک، m(s,t+1) را توصیف می‌کند.

تکامل جمعیت در الگوریتم ژنتیک وابسته به مرحله‌های انتخاب، جفت‌گیری و جهش است. بیایید فعلاً فقط تأثیر مرحله‌ی انتخاب را در نظر بگیریم. فرض کنید که f(h) میزان تناسب رشته بیت h، $\bar{f}(t)$ متوسط تناسب تمامی رشته بیت‌های جمعیت در زمان t ، n تعداد افراد جمعیت، $\hat{u}(s,t)$ متوسط تناسب تمامی اعضای الگوی s باشد، و می‌دانیم که h هم عضو الگوی s است و هم در زمان t در جمعیت حضور دارد $h \in s \cap p_t$.

حال می‌خواهیم مقدار m(s,t+1) را پیش‌بینی کنیم، این پیش‌بینی را با امید مقدار مذکور نشان می‌دهیم: E[m(s,t+1)] . با استفاده از رابطه‌ی تابع توزیع احتمال مطرح شده در جدول ۹,۱ می‌توانیم مقدار E[m(s,t+1)] را محاسبه کنیم. رابطه‌ی ۹,۱ را می‌توان با فرضیاتی که کردیم به شکل زیر بازنویسی کرد:

$$\Pr(h) = \frac{f(h)}{\sum_{i=1}^{n} f(h_i)}$$

$$= \frac{f(h)}{n\bar{f}(h)}$$

حال اگر h را یکی از اعضای جمعیت جدید در نظر بگیریم طبق تابع توزیع احتمال، احتمال اینکه عضوی از s را انتخاب کنیم را خواهیم داشت:

$$\Pr(h \in s) = \sum_{h \in s \cap p_t} \frac{f(h)}{n\bar{f}(t)}$$

$$= \frac{\hat{u}(s,t)}{n\bar{f}(h)} m(s,t) \qquad (9.2)$$

در نتیجه‌گیری دوم از این حقیقت استفاده کردیم که:

$$\hat{u}(s,t) = \frac{\sum_{h \in s \cap p_t} f(h)}{m(s,t)}$$

رابطه‌ی ۹,۲ احتمال این را نشان می‌دهد که فرضیه‌ی انتخاب شده توسط الگوریتم ژنتیک نمونه‌ای از s باشد. چون n انتخاب مستقل از هم صورت می‌گیرد امید تعداد اعضای انتخاب شده n برابر احتمال انتخاب هر یک از اعضا خواهد بود و خواهیم داشت:

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(h)} m(s,t) \qquad (9.3)$$

رابطه‌ی ۹,۳ نشان می‌دهد که امید تعداد نمونه‌های الگوی s در نسل t+1 با متوسط تناسب نمونه‌های الگوی s، $\hat{u}(s,t)$ رابطه‌ی مستقیم و با متوسط تناسب تمامی فرضیه‌ها در زمان t نسبت عکس دارد. پس می‌توانیم انتظار داشته باشیم که الگوهایی که متوسط تناسبشان بالای

متوسط تناسب کل است در نسل‌های بعدی نمونه‌های بیشتری را به خود اختصاص خواهند داد. اگر به الگوریتم ژنتیک بـه نگـاه جسـتجویی در میان الگوها، همزمان با جستجو در میان افراد برای پیدا کردن متناسب‌ترین‌ها نگاه کنیم، رابطه‌ی ۹,۳ نشان می‌دهد که در طی زمان الگوهـایی که تناسب بیشتری دارند رشد بیشتری خواهند داشت.

در عبارت بالا فقط مرحله‌ی انتخاب را در نظر گرفتیم، در حالی دو مرحله‌ی تولیدمثل و جهش نیز باید در نظر گرفته شوند. تئوری الگو فقـط اثـر منفی احتمالی این اعمال ژنتیکی را در نظر می‌گیرد (برای مثال جهش ممکن است تعداد نمونه‌های موجـود از الگـوی s را کـاهش دهـد)، و در تولیدمثل نیز فقط تولیدمثل تک نقطه‌ای را در نظر می‌گیرد. فرم کامل نظریه‌ی الگو کران پایینی بـرای امیـد تعـداد نمونـه‌های الگـوی s بیـان می‌کند:

$$E[m(s,t+1)] = \frac{\hat{u}(s,t)}{\bar{f}(h)} m(s,t)(1 - \frac{p_c d(s)}{l-1})(1-p_m)^{o(s)} \qquad (9.4)$$

در این رابطه p_c احتمال این است که عمل تولیدمثل تک نقطه‌ای به هر فرد دلخواهی اعمال شود p_m نیز احتمال این است که بیتی از فـردی دلخواه جهش کند. o(s) در این رابطه بیت‌های معلوم[۲۱] الگوی s است، در شمارش این بیت‌ها فقط بیت‌های ۰ و ۱ شمرده می‌شوند و بیت‌های * شمرده نمی‌شوند. d(s) نیز در این رابطه فاصله‌ی بیت چپ‌ترین و راست‌ترین بیت معلوم s است. L نیز طول هر رشته بیت در جمعیت است. توجه می‌کنید که قسمت اول رابطه‌ی ۹,۴ همان رابطه‌ی ۹,۳ است که تأثیر مرحله‌ی انتخاب بر تئوری الگـو اسـت. قسـمت بعـدی رابطـه اثـر تولیدمثل تک نقطه‌ای است، در کل این قسمت احتمال اینکه فرزندان هر نمونه‌ی بعد از تولیدمثل در s باشند را مشخص می‌کند. و قسمت آخر رابطه نیز اثر مرحله‌ی جهش است، این قسمت نیز احتمال اینکه بعد از جهش هنوز نمونه عضو s باشد را مشخص می‌کند. توجـه داریـد کـه دو عمل تولیدمثل تک نقطه‌ای و جهش تعداد بیت‌های معلوم الگو o(s) و فاصله‌ی بین بیت‌های معلوم d(s) را افزایش می‌دهند. بنابراین تئـوری الگو به این نتیجه‌گیری می‌رسد که در کل، الگوهای متناسب‌تر بیشتر رشد خواهند کرد، مخصوصاً الگوهایی که تعداد بیـت معلـوم کمـی دارنـد (تعداد * هایشان بالاست)، و الگوهایی که این بیت‌های معلوم به هم نزدیک‌ترند.

تئوری الگو شاید کلی‌ترین توصیف از تکامل در الگوریتم ژنتیک‌ها باشد. تنها ضعف این تئوری این است که در آن اثر مثبت احتمالی تولیدمثل و جهش یک چیز نادیده گرفته شده است. اخیراً نظریه‌های زیادی در مورد تکامل ارائه شده کـه بعضـی از آن‌هـا بـر پایـه‌ی مـدل‌های زنجیـروار مارکوف[۲۲] و بعضی دیگر بر پایه‌ی مـدل‌های مکانیزم‌هـای آمـاری هسـتند. بـرای اطلاعـات بیشـتر بـه (Whitley and Vose 1995) و (Mitchell 1996) مراجعه کنید.

۹,۵ برنامه‌نویسی ژنتیک

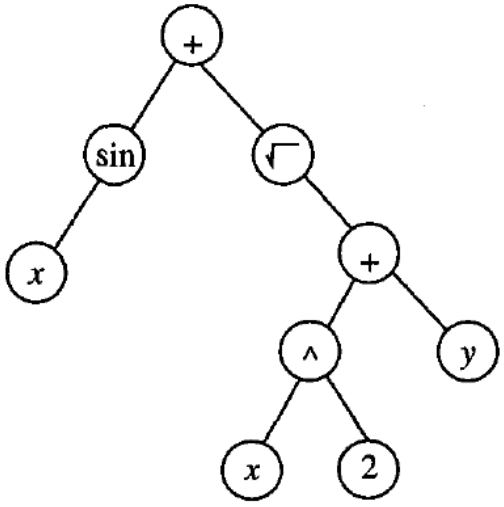
برنامه‌نویسی ژنتیک[۲۳] نوعی از محاسبات تکاملی است که در آن اعضای جمعیت‌ها به جای رشته بیت‌ها برنامه‌های کامپیوتری هستند. (Koza 1992) روش برنامه‌نویسی ژنتیک را توصیف کرده و دسته‌ی بزرگی از برنامه‌های ساده را که توسط GP می‌توان یاد گرفت را معرفی می‌کند.

---

[۲۱] defined bits
[۲۲] Markov chain models
[۲۳] genetic programming

۹,۵,۱ نمایش برنامه‌ها

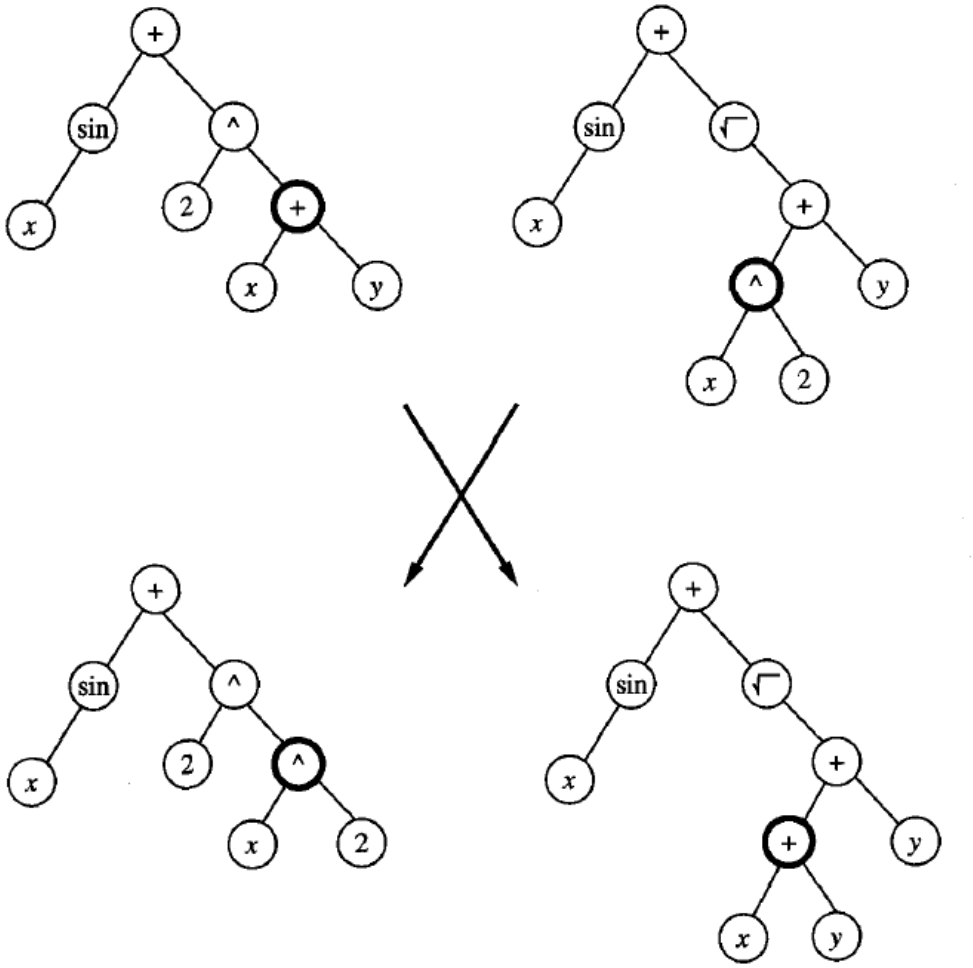
برنامه‌هایی که در GP مورد بحث قرار می‌گیرند معمولاً به صورت درخت‌هایی نمایش داده می‌شوند. هر تابع توسط یک گره در درخت و هر مقدار توسط یک یال مشخص می‌شود. برای مثال، شکل ۹,۱ تابع $\sin(x) + \sqrt{x^2 + y}$ را نشان می‌دهد. برای استفاده از برنامه‌نویسی ژنتیک باید ابتدا توابع پایه‌ای (مثل sin، cos، $\sqrt{}$، +، -، exp و ...) و ترمینال‌ها[۳۴] (مثل x، y، اعداد ثابت، و...) را مشخص کرد. برنامه‌نویسی ژنتیک با استفاده از محاسبات تکاملی جستجویی در فضای بزرگ فرضیه‌های که برنامه‌های ساخته شده توسط توابع پایه است را فراهم می‌کند.



شکل ۹,۱ درخت نمایش برنامه‌ها در برنامه‌نویسی ژنتیک.
در برنامه‌نویسی ژنتیک برنامه‌های دلخواه با درخت‌های متناسبشان نشان داده می‌شوند.

درست مشابه الگوریتم‌های ژنتیک در قالب کلی برنامه‌نویسی ژنتیک نیز جمعیت‌ها (این بار به شکل برنامه‌های درختی) حضور دارند. در هر تکرار حلقه‌ی اصلی، نسلی جدید از افراد در سه مرحله‌ی انتخاب، تولیدمثل و جهش ایجاد می‌شوند. تناسب هر برنامه‌ی جمعیت نیز با اجرای آن برای چند نمونه‌ی آموزشی به دست می‌آید. اعمال تولیدمثل نیز با انتخاب و عوض کردن جای دو زیرشاخه‌ی درخت برنامه‌های والد انجام می‌گیرد. شکل ۹,۲ یک تولیدمثل را نشان می‌دهد.

[۳۴] terminal



شکل ۹,۲ عمل تولیدمثل برای دو درخت والد (بالای شکل).
نقاط تولیدمثل (گره‌های نشان داده شده) به تصادف انتخاب می‌شوند. و سپس زیرشاخه‌ها جای خود را عوض می‌کنند تا فرزندان (پایین شکل) شکل بگیرند.

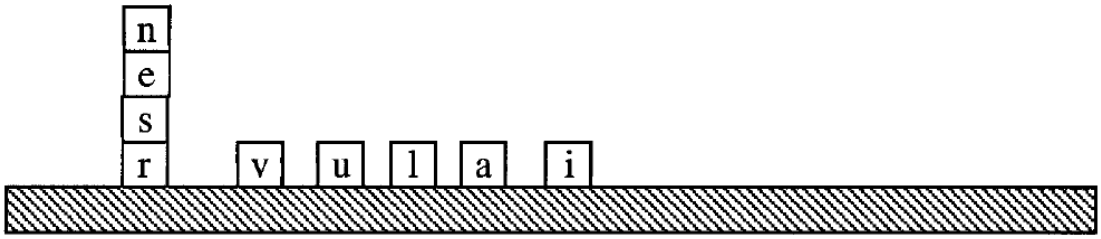
(Koza 1992) مجموعه‌ای GP را در تعدادی از مسائل به کار برد. در آزمایش‌های وی 10% از جمعیت فعلی با توجه به تناسبشان به صورت احتمالی مستقیماً به جمعیت نسل بعد انتقال داده می‌شدند. بقیه‌ی جمعیت نسل بعد از طریق عمل تولیدمثل جفت برنامه‌های نسل فعلی، که دوباره به صورت احتمالی و با توجه به تناسبشان انتخاب شده بودند، ایجاد می‌شد. در این آزمایش‌ها از عمل جهش استفاده نشده است.

۹,۵,۲ یک مثال

یکی از مثال‌های ارائه شده توسط (Koza 1992) درباره‌ی یادگیری الگوریتمی برای روی هم چیدن مکعب‌های شکل ۹,۳ بود. هدف پیدا کردن الگوریتمی کلی است که بدون تأثیر چینش اولیه‌ی مکعب‌ها، آن‌ها را طوری روی هم بچینند که در آخر در یک ستون کلمه‌ی "universal" را نمایش دهند. در هر حرکت تنها می‌توان یک مکعب را جابجا کرد. در کل دو حرکت مجاز وجود دارد، یکی اینکه یک مکعب را از سطح زمین به بالای ستونی ببریم و دیگری اینکه مکعب بالایی ستون را به زمین منتقل کنیم.

مثل اکثر مسئله‌های برنامه‌نویسی ژنتیک، انتخاب نحوه‌ی نمایش مسئله نقش بسیار مهمی در آسان شدن حل آن دارد. در نمایشی که Koza برای مسئله انتخاب کرد، سه ترمینال زیر را انتخاب کرد:

- CS (ستون فعلی)، که نشان‌دهنده‌ی حرف مکعب بالای ستون است، در صورت عدم وجود ستون فعلی F خواهد بود.
- TB (بالاترین مکعب درست)، که نشان‌دهنده‌ی حرف مکعب بالای ستون درست است، در چنین ستونی تمامی حروف در ترتیب درست قرار دارند.
- NN (نیاز بعدی)، حرفی است که در ترتیب درست بالای ستون TB باید قرار بگیرد تا کلمه‌ی "universal" را تشکیل دهد، اگر تعداد مکعب‌ها کافی نبود مقدار آن F خواهد بود.



شکل ۹,۳ مسئله چینش مکعب. هدف از برنامه‌نویسی ژنتیک پیدا کردن برنامه‌ای است که بدون توجه به ترتیب اولیه مکعب‌ها آن‌ها را به فرمی بچیند که کلمه "universal" را ایجاد کنند. مجموعه‌ای از ۱۶۶ حالت اولیه برای تشخیص تناسب برنامه به کار می‌رود، (Koza 1992).

همان‌طور که دیده می‌شود، این انتخاب ترمینال‌های ورودی یک نمایش طبیعی برای توصیف این مسئله را ایجاد می‌کند. فرض کنید، در مقابل، به جای این ورودی، ورودی x و y تمامی مکعب‌های موجود است.

علاوه بر این ترمینال‌ها در برنامه‌های استفاده شده این توابع پایه‌ای به کار می‌روند:

- (MS x) (حرکت به روی ستون) اگر مکعب x روی زمین باشد این عمل آن را به بالای ستون می‌برد و مقدار T را بر می‌گرداند. در غیر این صورت عملی صورت نمی‌گیرد و مقدار F بر گردانده می‌شود.
- (EQ x y) (تساوی) اگر x و y مساوی باشند مقدار T برگردانده می‌شود، در غیر این صورت F برگردانده می‌شود.
- (NOT x) اگر x=F، مقدار T را برمی‌گرداند و اگر x=T مقدار F را بر می‌گرداند.
- (DU x y) ("Do Until") دستور x را تا T شدن y تکرار خواهد کرد.

برای اینکه سیستم بتواند تناسب برنامه‌های تولیدی را ارزیابی کند، Koza، ۱۶۶ نمونه‌ی آموزشی از چینش اولیه‌ی مختلف مکعب‌ها با سختی‌های متفاوت ایجاد کرد. تناسب هر یک از برنامه‌ها تعداد نمونه‌های آموزشی حل شده توسط برنامه تلقی می‌شد. در ابتدا جمعیتی با ۳۰۰ برنامه‌ی به تصادف ایجاد می‌شود. بعد از ده نسل سیستم به برنامه‌ی زیر رسید که تمامی نمونه‌های آموزشی را حل می‌کرد:

(EQ (DU (MT CS)(NOT CS)) (DU (MS NN)(NOT NN)))

توجه دارید که این برنامه از دو دستور DU یا همان "Do Until" استفاده کرده است. در دسته حرکت اول تمامی مکعب‌ها به روی زمین انتقال داده می‌شوند تا مجموعه‌ی ستون‌ها خالی شود. در دسته حرکت دوم پشت سر هم نیاز بعدی بر روی ستون درست قرار می‌گیرد. نقش دستور EQ اول در اینجا ایجاد قاعده‌ای دستوری برای ترکیب دو حلقه‌ی "Do Until" است.

جای تعجب است که این برنامه‌نویسی ژنتیک بعد تنها چند نسل به برنامه‌ای می‌رسد که تمامی ۱۶۶ نمونه‌ی آموزشی را حل می‌کند. البته قدرت سیستم برای حل مسئله رابطه‌ی بسیار نزدیکی با ترمینال‌ها و توابع پایه‌ای و نمونه‌های آموزشی دارد.

۹,۵,۳ نکاتی درباره‌ی برنامه‌نویسی ژنتیک

همان‌طور که در مثال بالا آورده شد، برنامه‌نویسی ژنتیک از الگوریتم‌های ژنتیک به تکامل برای برنامه‌های کامل کامپیوتری رسید. برخلاف اندازه‌ی بزرگ فضای فرضیه‌ای موجود که جستجو را سخت‌تر می‌کند، برنامه‌نویسی ژنتیک اثبات کرده است که می‌تواند نتایج خیره‌کننده‌ای در بعضی کاربردها بدهد. مقایسه‌ای بین برنامه‌نویسی ژنتیک و دیگر متدهای جستجوی فضای برنامه‌های کامپیوتری، مثل hillclimbing و simulated annealing در کتاب (O'Reilly and Oppacher 1994) انجام شده است.

با این وجود که مثالی که در بالا آمده کمی ساده بود، (Koza 1996) و دیگر محققان کاربرد برنامه‌نویسی ژنتیک را در کاربردهای پیچیده‌تر مثل طراحی مدارهای فیلتر الکتریکی و دسته‌بندی مولکول‌های پروتئین‌ها نشان داد. مسئله‌ی طراحی فیلتر مدار نمونه‌ای از مسئله‌های نسبتاً پیچیده است. در این مسئله، برنامه‌ها تکامل می‌یابند تا بتوانند از مدارات پایه‌ای[۲۵] مداری پیشرفته (مثل فیلتر) طراحی کنند. در این مسئله توابع پایه‌ای توابعی هستند که مدارات پایه‌ای را با اضافه یا کم کردن عناصر مداری و اتصالات تغییر می‌دهند. تناسب هر برنامه توسط برنامه‌های شبیه‌ساز (مثل SPICE) و از طریق مقایسه‌ی خروجی و خروجی مطلوب فیلتر محاسبه می‌شود. به عبارت دیگر امتیاز تناسب مجموع اندازه‌های تمام خطاهای بین مطلوب و خروجی مدار در ۱۰۱ فرکانس متفاوت است. در این مسئله در هر نسل جمعیتی با ۶۴۰۰۰۰ عضو ایجاد می‌شد که ۱۰٪ از جمعیت قبلی، ۸۹٪ حاصل تولیدمثل و ۱٪ نیز حاصل جهش بودند. سیستم بر روی یک پردازنده‌ی ۶۴ پایه‌ای موازی اجرا می‌شد. در نسل تصادفی مدارات آن‌قدر بی‌محتوا بودند که SPICE نمی‌توانست ۹۸٪ شان را شبیه‌سازی کند. در نسل بعدی یا همان نسل اول این میزان به ۸۴,۹٪ و در نسل دوم به ۷۵٪ کاهش یافت. در نسل‌های موفق این مقدار حتی تا ۹,۶٪ نیز کاهش یافت. امتیاز تناسب بهترین مدار در نسل اولیه ۱۵۹ بود، بعد از ۲۰ نسل این میزان به ۳۹ و بعد از ۱۳۷ نسل به ۰,۸ کاهش یافت. بهترین مدار بعد از ۱۳۷ نسل ایجاد شد که رفتاری بسیار شبیه به رفتار مطلوب داشت.

در اکثر موارد، کارایی برنامه‌نویسی ژنتیک مستقیماً به طرز نمایش و انتخاب تابع تناسب وابسته است. به همین دلیل، قسمت عمده‌ای از تحقیقات به چگونگی انتخاب خودکار توابع پایه و الحاق آن‌ها برای بهبود توابع پایه‌ای اصلی می‌پردازد، این کار به سیستم اجازه می‌دهند که به صورت پویا توابع پایه‌ی برنامه‌ها را تغییر دهد. برای اطلاعات بیشتر به (Koza 1994) مراجعه کنید.

۹,٦ مدل‌های تکاملی و یادگیری

در بسیاری از سیستم‌های طبیعی، بسیاری از افراد در طول زندگی‌شان تطابق‌های مهمی را یاد می‌گیرند. در همین زمان، فرایندهای زیستی و اجتماعی به گونه‌شان این امکان را می‌دهد تا در طول نسل‌ها تطابق پیدا کنند. یکی از سؤال‌های بسیار جالب در مورد سیستم‌های تکاملی این است که "رابطه‌ی بین یادگیری در طول عمر یک فرد با یادگیری در طول چندین نسل توسط تکامل چیست؟"

۹,٦,۱ تکامل لامارکی

لامارک (Lamarck) محققی بود که در سال‌های آخر قرن نوزدهم می‌زیست. وی اعتقاد داشت که تکامل در طول نسل‌ها مستقیماً به تجربه‌های فردی در طول عمر وابسته است. در کل، وی اعتقاد داشت که تجارب یک فرد مستقیماً بر چینش ژنتیکی فرزندانش تأثیر می‌گذارد: اگر فردی در طول زندگی‌اش بیاموزد که از غذایی سمی پرهیز کند می‌تواند این آموزش را از طریق ژنتیکش به فرزندانش منتقل کند، پس

---

[۲۵] simple fixed seed circuit

فرزندانش دیگر نیازی به یاد گرفتن چنین چیزی ندارند. این حدس بسیار جذاب است، زیرا که فرایند تکامل مؤثرتر خواهد شد به جای اینکه فقط یک آزمون‌وخطا باشد که تجارب فردی در طول عمر را فراموش کند (مثل نمونه‌ای که در الگوریتم‌های ژنتیک و برنامه‌نویسی ژنتیک بود). با وجود تمامی جذابیت‌های این نظریه، محققین عصر حاضر مدارک انکار ناشدنی‌ای برای رد کردن مدل لامارک دارند. نظریه‌ی پذیرفته شده‌ی فعلی این است که نقشه‌ی ژنتیکی هر فرد، در واقع، هیچ تأثیری از تجارب طول زندگی والدینش نمی‌پذیرد. برخلاف این حقیقت زیستی، تحقیقات جدید کامپیوتری نشان داده که فرایند لامارکی گاهی می‌تواند کارایی الگوریتم‌های ژنتیکی کامپیوتری را بهبود ببخشد (برای اطلاعات بیشتر به Grefenstette 1991 و Ackley and Littman 1994 و Hart and Belew 1995 مراجعه کنید).

۹,٦,۲ اثر بالدوین

با وجود اینکه مدل تکاملی لامارکی برای تکامل زیستی رد شد، مکانیزم‌های دیگری پیشنهاد شده که در آن‌ها یادگیری‌های فردی مسیر تکامل را می‌تواند عوض کند. یکی از این مکانیزم‌ها اثر بالدوین[۲۶] است که نامش نیز از بالدوین (J. M. Baldwin 1896)، اولین کسی که این نظر را ارائه داد، گرفته شده است. اثر بالدوین بر پایه‌ی مشاهدات زیر نتیجه‌گیری شده:

- اگر گونه‌ای در حال تکامل در یک محیط در حال تغییر باشد، تمایل تکامل به سمتی خواهد بود تا افراد در طول عمر خود قابلیت یادگیری بیشتری داشته باشند. برای مثال، با ظاهر شدن شکارچی جدید، افرادی که قابلیت یادگیری پرهیز از این شکارچی را دارند از افرادی که این قابلیت را ندارند موفق‌تر خواهند بود. پس قابلیت یادگیری به فرد اجازه می‌دهد تا جستجویی منطقه‌ای انجام دهد تا تناسبش را به حداکثر برساند. در مقابل، افرادی که این قابلیت فردی را ندارند و تناسبشان نیز توسط ژنتیکشان محدود شده در نقطه‌ی پایین‌تری نسبت به گروه اول قرار می‌گیرند.
- افرادی که می‌توانند بسیاری از آموزش‌های لازم را یاد بگیرند برای یادگیری کمتر به کد ژنتیکی‌شان متکی خواهند بود. نتیجه اینکه این افراد تعداد جمعیت‌هایی با گوناگونی ژنتیکی بیشتری را تشکیل می‌دهند و از قابلیت‌های فردی‌شان برای غلبه بر کمبودهای ژنتیکی استفاده می‌کنند. چنین جمعیت ژنتیکی گوناگونی می‌تواند باعث تکامل بیشتر ژنتیکی شوند. پس، قابلیت افراد برای یادگیری می‌تواند اثری غیرمستقیم بر افزایش سرعت تکامل کل جمعیت داشته باشد.

برای تصور فرض کنید که در محیط گونه‌ی خاصی تغییراتی جدید ایجاد می‌شود، مثلاً یک شکارچی اضافه می‌شود. چنین تغییری باعث می‌شود که فقط گونه‌هایی که قابلیت فردی پرهیز از شکارچی را دارند زنده بمانند. به تناسب میزان تطبیق‌پذیری فردی هر فرد در افزایش جمعیت، جمعیت می‌تواند انواع گوناگون‌تری از نمونه‌های ژنتیکی را در خود داشته باشد و سرعت فرایندهای تکامل را تسریع ببخشد . این افزایش سرعت تطبیق ممکن است باعث شود که گونه‌هایی به وجود بیایند که به طور ژنتیکی از شکارچی پرهیز کنند (مثلاً حسی غریزی برای پرهیز از بعضی مناطق). پس، اثر بالدوین مکانیزم‌های غیرمستقیمی را ایجاد می‌کند تا یادگیری‌های فردی نیز بر سیر تکامل تأثیر داشته باشند. با افزایش مقاومت[۲۷] و گوناگونی ژنتیکی در میان گونه‌ها، یادگیری‌های فردی سرعت تکامل را سریع‌تر خواهند کرد و شانس ایجاد گونه‌هایی که به طور ژنتیکی در محیط جدید برتر از گونه‌های قبلی هستند افزایش می‌یابد.

تلاش‌های بسیاری برای مطالعه‌ی اثر بالدوین در مدل‌های محاسباتی شده است. برای مثال، آزمایش‌های (Hinton and Nowlan 1987) بر روی نمونه‌ها ساده‌ای از شبکه‌های عصبی انجام شد، در این آزمایش وزن‌های بعضی شبکه‌ها در تمام طول زندگی‌اش ثابت بود در

---

[۲۶] Baldwin effect

[۲۷] survivability

حالی که بعضی دیگر قابل‌آموزش بودند. نقشه‌ی ژنتیکی هر فرد مشخص می‌کرد که شبکه آموزش پذیر باشد یا نه. در این آزمایش، در افـرادی که نمی‌توانستند یاد بگیرند، بعد از نسل‌ها هیچ تکاملی در تناسب افراد ایجاد نشد، اما در افرادی که می‌توانستند یاد بگیرند تناسـب جمعیـت بـه شدت افزایش یافت. در نسل‌های ابتدایی تکامل جمعیت تعداد نسبی افرادی که می‌توانستند یاد بگیرند بسیار زیاد بود. با این وجود با ادامه یافتن تکامل تعداد شبکه‌هایی که وزن‌های ثابت داشتند و درست کار می‌کردند تمایل به افزایش یافت و جمعیت به سمت قابلیت ژنتیکی میل کرد تـا اینکه بر توانایی‌های فردی اتکا بزند. تحقیقات دیگری نیز در مورد اثر بالدوین در الگوریتم ژنتیک‌هـا توسـط افـرادی مثـل (Belew 1990)، (Harvey 1993)، و (French and Messinger 1994) انجام شده است. بررسی کاملی نیز در (Mitchell 1996) انجـام شـده است. قسمت خاصی از مجله‌ی (journal Evolutionary Computation) نیز در این باره مطالب مفیدی دارد (Turney 1997).

۹,۷ موازی‌سازی الگوریتم‌های ژنتیک

الگوریتم‌های ژنتیک ذاتاً مناسب پیاده‌سازی موازی‌اند، و روش‌های متعددی برای موازی‌سـازی آن‌هـا پیـدا شـده اسـت. روش‌هـای coarse grain برای موازی‌سازی و تقسیم جمعیت به گروه‌های مجزا افراد به نام بخش[۲۸] به کار می‌روند. هر بخش بـه گره‌هـای محاسـبه‌گر مختلـف ارجاع می‌شود تا جستجویی استاندارد بر اساس الگوریتم ژنتیک در آنجا انجام بپذیرد. ارتباطات و جفت‌گیری مشـترک بـین بخش‌هـا نیـز اتفـاق می‌افتند اما احتمال آن از جفت‌گیری درون بخش خیلی کمتر است. انتقال بین بخش‌ها نیز توسط فرایند مهاجرت[۲۹] اتفـاق مـی‌افتـد، کـه در آن افرادی از یک بخش به بخش یا بخش‌های دیگری کپی یا انتقال داده می‌شوند. این فرایندها از جفت‌گیری و مهـاجرت بـین زیـر جمعیت‌هـای موجود در گونه‌های زیستی الهام گرفته شده است. یکی از فواید این گونه تقسیم نسبت به سیستم‌های غیر موازی کاهش تـراکم آن‌هاسـت، در مشکل تراکم به خاطر ظهور زودهنگام یکی از گونه‌های برتر تمامی نمونه‌ها به سمت آن متمایل می‌شوند و جامعه توسط این گونه پر می‌شـود. نمونه‌ی الگوریتم ژنتیک‌هایی که از موازی‌سازی coarse-grain بهره مـی‌برنـد در (Tanese 1989) و (Cohoon 1987) و... آمـده است.

در نقطه‌ی مقابل موازی‌سازی coarse-grain؛ موازی‌سازی fine-grain است که معمولاً یک پردازنده برای هـر فـرد در جمعیـت در نظـر می‌گیرد. و عمل جفت‌گیری فقط در افراد همسایه رخ می‌دهد. چندین نوع دیگری از تعریف همسایگی نیز ارائه شده است. در بعضـی تعریف‌هـا هر فرد فقط دو همسایه دارد و در بعضی دیگر نیز هر فرد با محدوده‌ی دایره‌ای خاص اطراف خود همسایه است. نمونه‌های چنـین سیسـتم‌هایی در (Spiessens and Manderick 1991) آورده شده. منتخبی از تحقیقات الگوریتم ژنتیک نیز در (Stender 1993) آمده است.

۹,۸ خلاصه و منابع برای مطالعه‌ی بیشتر

نکات اصلی این فصل شامل موارد زیر است:

- الگوریتم‌های ژنتیک (GA) جستجویی تصادفی، موازی و hill-climbing برای پیدا کردن فرضیه‌هایی که تـابع از پـیش تعریـف شده‌ی تناسب را بهینه می‌کنند انجام می‌دهند.

---

[۲۸] deme

[۲۹] migration process

- این جستجوی الگوریتم‌های ژنتیک بر اساس تشابه با تکامل بیولوژیکی انجام می‌شود. جمعیت‌های گوناگون فرضیه‌های رقیب ایجاد می‌شوند. در هر حلقه، متناسب‌ترین اعضای جمعیت انتخاب شده و فرزندان از آن‌ها تولید می‌شوند، این فرزندان جایگزین اعضای کم تناسب تر جمعیت خواهند شد. فرضیه‌ها به صورت رشته بیت‌ها کد می‌شوند و توسط اعمال تولیدمثل ترکیب شده و یا توسط جهش تغییر می‌یابند.
- الگوریتم‌های ژنتیک کار یادگیری را به عنوان نوعی بهینه‌سازی مطرح می‌کنند. در کل، عمل یادگیری در این دیدگاه پیدا کردن فرضیه‌های بهینه، برای تابع از پیش تعیین شده‌ی تناسب، است. این دیدگاه باعث می‌شود که بتوان دیگر روش‌های بهینه‌سازی مثل simulated annealing را نیز در یادگیری ماشین به کار برد.
- الگوریتم‌های ژنتیک معمولاً در مسائل بهینه‌سازی خارج محدوده‌ی یادگیری ماشین، مثل مسائل بهینه‌سازی طراحی به کار رفته‌اند. در یادگیری ماشین، الگوریتم‌های ژنتیک معمولاً برای کارهای یادگیری پیچیده (یادگیری دسته قوانین کنترل ربات و یادگیری برنامه‌های کامپیوتری) به کار می‌روند، در این مسائل هدف بهینه‌سازی تابع غیرصریحی از فرضیه‌هاست (مجموعه قوانین ربات را به طور صحیح کنترل کند).
- برنامه‌نویسی ژنتیک نسخه‌ای از الگوریتم‌های ژنتیک است که در آن فرضیه‌هایی که توسط کامپیوترها دست‌کاری می‌شوند، برنامه‌های کامپیوتری هستند، بجای رشته بیت‌ها. اعمالی چون تولیدمثل و جهش به برنامه‌ها بجای رشته بیت‌ها تعمیم داده می‌شود. برنامه‌نویسی ژنتیک اثبات کرده که می‌تواند برنامه‌هایی برای کارهایی نظیر کنترل ربات (1992) Koza و تشخیص اشیا در تصاویر را انجام دهد (1994) Teller and Veloso.

روش‌های محاسباتی مبتنی بر تکامل از روزهای اولیه‌ی علم کامپیوتر مورد بررسی قرار گرفتند (Box 1957 and Bledsoe 1961). روش‌های تکاملی مختلف بسیاری در دهه‌ی ۱۹۶۰ معرفی شد و در آن زمان مورد تحقیق بیشتر قرار گرفت. استراتژی‌های تکاملی، توسط (Rechengerg 1965,1973) برای بهینه کردن پارامترهای عددی در طراحی مهندسی طراحی شده و با کارهای Schwefel (1975, 1977, 1995) و دیگران ادامه یافت. برنامه‌نویسی تکاملی توسط Folgel, Owens, and Walsh (1966) به عنوان متدی برای ساخت ماشین‌های finite-state طراحی شد و توسط محققان (Fogel and Atmar 1993) عددی ادامه یافت. الگوریتم‌های ژنتیک، معرفی شده توسط Holland (1962, 1975) شامل مفهوم حفظ جمعیتی بزرگ از افراد و تأکید بر تولیدمثل به عنوان عملگر کلیدی در چنین سیستم‌هایی می‌شد. الگوریتم‌های ژنتیک، معرفی شده توسط Koza (1992) استراتژی جستجوی الگوریتم‌های ژنتیک را به فرضیه‌ها برنامه‌های کامپیوتری اعمال می‌کند. با کاهش قیمت کامپیوترها و افزایش سرعتشان، علاقه به روش‌های تکاملی بیشتر می‌شود.

یکی از روش‌های استفاده از الگوریتم‌های ژنتیک یادگیر دسته قوانین است که توسط K. DeJong و دانشجویانش در دانشگاه Pittsburg است (Smith 1980). در این روش، هر مجموعه قوانین یکی از اعضای جمعیت رقابتی فرضیه‌هاست، همان‌طور که در سیستم GABIL در این فصل نیز توضیح داده شد. روش دیگری که در دانشگاه Michigan توسط Holland و دانشجویانش (Holland 1986) ایجاد شده، روشی است که در آن هر قانون عضوی از جمعیت است و جمعیت خود یک دسته قانون است. تصور زیستی از نقش جهش، جفت‌گیری، جفت‌گیری بین نژادی[۳۰] و انتخاب در تکامل در Wright (1977) آورده شده است.

Mitchell (1996) و Goldberg (1989) دو کتاب مربوطه‌ی الگوریتم‌های ژنتیک هستند. Forrest (1993) نیز کتابی است که نگاه کلی‌ای از مسائل الگوریتم‌های ژنتیک را بررسی می‌کند، Goldberg (1994) نیز نگاه کلی‌ای از چندین کاربرد جدید الگوریتم‌های

---

[۳۰] cross-breeding

ژنتیک را در بر دارد. کتاب Koza (1992) نیز درباره‌ی برنامه‌نویسی ژنتیک منبع استاندارد تعمیم الگوریتم‌های ژنتیک برای تغییر برنامه‌های کامپیوتری است. کنفرانس‌های اولیه که در آن نتایج اولیه انتشار می‌شود کنفرانس International Conference on Genetic Algorithms است. دیگر کنفرانس‌های مربوطه شامل Conference on Simulation of Adaptive Behavior و the International Conference on Artificial Neural Networks and Genetic Algorithms و IEEE International Conference on Evolutionary Computation هستند. کنفرانس سالانه‌ای نیز درباره‌ی برنامه‌نویسی ژنتیک برگزار می‌شود (Koza et al. 1996b). The Evolutionary Computation Journal نیز یکی از منابع اخیر نتایج تحقیقات در این زمینه است. بسیاری از قسمت‌های the journal Machine Learning نیز به الگوریتم‌های ژنتیک اختصاص یافته است.

تمرینات

۹,۱ الگوریتم ژنتیکی طراحی کنید که قوانین دسته‌بندی عطفی را برای مفهوم PlayTennis مطرح شده در فصل ۳ را یاد بگیرد. رشته‌های کد برای فرضیه‌ها و عملگرهای ژنتیک را دقیقاً مشخص کنید.

۹,۲ نمونه‌ی ساده‌ای از الگوریتم ژنتیک تمرین ۹,۱ را پیاده‌سازی کنید. الگوریتم را با اندازه جمعیت‌های مختلف p، نسبت جایگزینی‌های مختلف r و ضریب جهش‌های مختلف m امتحان کنید.

۹,۳ برنامه‌ی ایجاد شده توسط برنامه‌نویسی ژنتیک در بخش ۹,۵,۲ را به صورت درخت پیدا کنید. عملگر تولیدمثل برنامه‌نویسی ژنتیک را می‌توان با استفاده از یک درخت به عنوان هر دو والد در نظر گرفت.

۹,۴ استفاده از برنامه‌نویسی ژنتیک را در پیدا کردن بردار وزن متناسب با یک شبکه‌ی عصبی مصنوعی را در نظر بگیرید (در کل شبکه‌ای تک‌سویه مشابه مواردی که در فصل ۴ با backpropagation آموزش دادیم). شبکه‌ای ۱×۲×۳ لایه‌ای و تک‌سویه را در نظر بگیرید. کد سازی‌ای برای وزن‌های شبکه ارائه دهید و مجموعه‌ای از اعمال ژنتیک مناسب روی این کدها را توصیف کنید. یک مزیت و یک مشکل استفاده از ژنتیک بجای backpropagation را برای آموزش شبکه‌های عصبی بیان کنید.

فرهنگ لغات تخصصی فصل (فارسی به انگلیسی)

اثر بالدوین	Baldwin effect
اشتراک تناسب	fitness sharing
انتخاب رتبه‌ای	rank selection
انتخاب مسابقه‌ای	tournament selection
انتخاب نسبی تناسبی	fitness proportionate selection
برنامه‌ریزی مغازه‌داری	job-shop scheduling
برنامه‌نویسی ژنتیک	genetic programming
بیت‌های معلوم	defined bits
تابع تناسب	fitness function
تراکم	Crowding

ترمینال	Terminal
تناسب	Fitness
مقاومت	Survivability
تولیدمثل	crossover, offspring
تولیدمثل تک نقطه‌ای	single-point crossover
تولیدمثل یکنواخت	uniform crossover
تئوری الگو	Schema Theorem
جستجویی ستونی	beam search
جمعیت	Population
جهش	Mutation
دسته قوانین	sets of rules, classification rules
زیررشته	Substring
زیرگونه‌ها	Subspecies
طراحی مدار	circuit layout
قانون‌های فصلی گزاره‌ای	disjunctive set of propositional rules
شرط	Constraint
الگو	Schema, pattern
الگوریتم‌های ژنتیک	Genetic algorithms
محاسبات تکاملی	evolutionary computation
مدارات پایه‌ای	simple fixed seed circuit
مدل‌های زنجیروار مارکوف	Markov chain models
مشکل تراکم	crowding problem
نشانه‌های نمادین	symbolic expressions
نقاب تولیدمثل	crossover mask
نقشه‌ی ژنتیکی	Genetic makeup
نمایش نمادین	symbolic expressions
یادگیری ماشین	machine learning