دانشگاه آزاد اسلامی واحد تبریز

نام درس: یادگیری ماشین

بخش: الگوریتم های ژنتیک

نام استاد: دکتر مسعود کارگر

MASOUDKARGAR.IR

الگوریتم ژنتیک

- الگوریتم ژنتیک روش یادگیری بر پایه تکامل بیولوژیک است.
- این روش در سال 1970 توسط John Holland معرفی گردید.
- این روشها با نام Evolutionary Algorithms نیز خوانده می‌شوند.

ایده کلی

- یک GA برای حل یک مسئله مجموعه بسیار بزرگی از راه‌حلهای ممکن را تولید می‌کند.
- هر یک از این راه‌حلها با استفاده از یک " تابع تناسب "مورد ارزیابی قرار می‌گیرد.
- آنگاه تعدادی از بهترین راه‌حلها باعث تولید راه‌حلهای جدیدی می‌شوند. که اینکار باعث تکامل راه‌حلها می‌گردد.
- بدین ترتیب فضای جستجو در جهتی تکامل پیدا می‌کند که به راه‌حل مطلوب برسد.
- در صورت انتخاب صحیح پارامترها، این روش می‌تواند بسیار موثر عمل نماید.

فضای فرضیه

- الگوریتم ژنتیک به جای جستجوی فرضیه‌های general-to specific و یا simple to complex فرضیه‌های جدید را با تغییر و ترکیب متوالی اجزا بهترین فرضیه‌های موجود بدست می‌آورد.
- در هر مرحله مجموعه‌ای از فرضیه‌ها که جمعیت (population)نامیده می‌شوند از طریق جایگزینی بخشی از جمعیت فعلی با فرزندانی که از بهترین فرضیه‌های موجود حاصل شده‌اند بدست می‌آید.

ویژگیها

- الگوریتم‌های ژنتیک در مسائلی که فضای جستجوی بزرگی داشته باشند می‌تواند بکار گرفته شود.
- همچنین در مسایلی با فضای فرضیه پیچیده که تاثیر اجزا آن در فرضیه کلی ناشناخته باشند می‌توان از GA برای جستجو استفاده نمود.
- برای Discrete Optimization بسیار مورد استفاده قرار می‌گیرد.
- الگوریتم‌های ژنتیک را می‌توان براحتی به صورت موازی اجرا نمود از این رو می‌توان کامپیوترهای ارزان قیمت‌تری را به صورت موازی مورد استفاده قرار داد.
- امکان به تله افتادن این الگوریتم در مینیمم محلی کمتر از سایر روشهاست.
- از لحاظ محاسباتی پرهزینه هستند.
- تضمینی برای رسیدن به جواب بهینه وجود ندارد.

Parallelization of Genetic Programming

- در سال 1999 شرکت Genetic Programming Inc. یک کامپیوتر موازی با 1000 گره هر یک شامل کامپیوترهای P2, 350 MHZ برای پیاده‌سازی روش‌های ژنتیک را مورد استفاده قرار داد.

کاربردها

- کاربرد الگوریتم‌های ژنتیک بسیار زیاد می‌باشد:
- Optimization,
- Automatic Programming,
- Machine Learning,
- Economics,
- Operations Research,
- Ecology,
- Studies Of Evolution And Learning, And
- Social Systems

زیر شاخه‌های EA

روش‌های EA به دو نوع مرتبط بـه هـم ولـی مجـزا دسـته‌بنـدی می‌شوند:

1. **Genetic Algorithms (GAs)**

در این روش راه‌حل یک مسئله به صورت یک Bit String نشان داده می‌شود.

2. **Genetic Programming (GP)**

این روش به تولید Expression Trees که در زبانهـای برنامـه نویسی مثل Lisp مورد استفاده هسـتند مـی‌پـردازد بـدین ترتیب می‌توان برنامه‌هائی ساخت که قابل اجرا باشند.

الگوریتم‌های ژنتیک

- روش متداول پیاده‌سازی الگوریتم ژنتیک بدین ترتیب است:
- استخری از فرضیه‌ها که Population نامیده می‌شود تولید و به طور متناوب با فرضیه‌های جدیدی جایگزین می‌گردد.
- در هر بار تکرارتمامی فرضیه‌ها با استفاده از یک تابع تناسب یا Fitness مورد ارزیابی قرار داده می‌شوند .آنگاه تعدادی از بهترین فرضیه‌ها با استفاده از یک تابع احتمال انتخاب شده و جمعیت جدید را تشکیل می‌دهند.
- تعدادی از این فرضیه‌های انتخاب شده به همان صورت مورد استفاده واقع شده و مابقی با استفاده از اپراتورهای ژنتیکی نظیر Crossover و Mutation برای تولید فرزندان بکار می‌روند.

پارامترهای GA

یک الگوریتم *GA* دارای پارامترهای زیر است:

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

- *Fitness*: تابعی برای ارزیابی یک فرضیه که مقداری عددی به هر فرضیه نسبت می‌دهد.
- *Fitness_threshold*: مقدار آستانه که شرط پایان را معین می‌کند.
- *p*: تعداد فرضیه‌هائی که باید در جمعیت در نظر گرفته شوند.
- *r*: درصدی از جمعیت که در هر مرحله توسط الگوریتم *Crossover* جایگزین می‌شوند.
- *m*: نرخ *Mutation*

GA($Fitness$, $Fitness\_threshold$, p, r, m)

Fitness: A function that assigns an evaluation score, given a hypothesis.

Fitness_threshold: A threshold specifying the termination criterion.

p: The number of hypotheses to be included in the population.

r: The fraction of the population to be replaced by Crossover at each step.

m: The mutation rate.

- *Initialize population:* $P \leftarrow$ Generate p hypotheses at random
- *Evaluate:* For each h in P, compute $Fitness(h)$
- While $[\max_h Fitness(h)] < Fitness\_threshold$ do

  Create a new generation, P_S:

  1. *Select:* Probabilistically select $(1 - r)p$ members of P to add to P_S. The probability $\Pr(h_i)$ of selecting hypothesis h_i from P is given by

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

  2. *Crossover:* Probabilistically select $\frac{r \cdot p}{2}$ pairs of hypotheses from P, according to $\Pr(h_i)$ given above. For each pair, $\langle h_1, h_2 \rangle$, produce two offspring by applying the Crossover operator. Add all offspring to P_s.
  3. *Mutate:* Choose m percent of the members of P_s with uniform probability. For each, invert one randomly selected bit in its representation.
  4. *Update:* $P \leftarrow P_s$.
  5. *Evaluate:* for each h in P, compute $Fitness(h)$
- Return the hypothesis from P that has the highest fitness.

الگورتیم

- *Initialize*: جمعیت را با تعداد p فرضیه به طور تصادفی مقداردهی اولیه کنید.
- *Evaluate*: برای هر فرضیه h در p مقدار تابع $Fitness(h)$ را محاسبه نمائید.
- تا زمانیکه $[max_h\ Fitness(h)] < Fitness\_threshold$ یک جمعیت جدید ایجاد کنید.
- فرضیه‌ای که دارای بیشترین مقدار *Fitness* است را برگردانید.

نحوه ایجاد جمعیت جدید

مراحل ایجاد یک جمعیت جدید به صورت زیر است:

1. *select*: تعداد $(1-r)p$ فرضیه از میان P انتخاب و به P_s اضافه کنید .احتمال انتخاب یک فرضیه h_i از میان P عبارت است از:

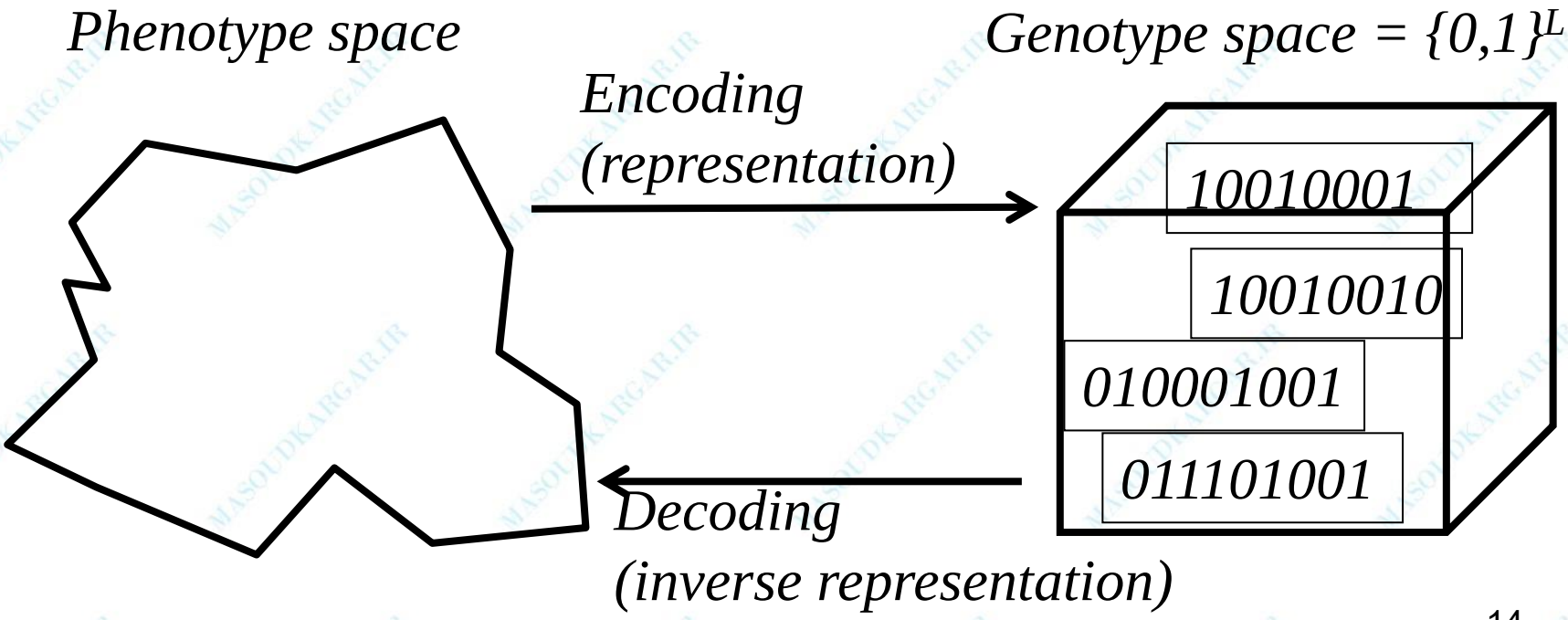
$$P(h_i) = Fitness\ (h_i) / \Sigma j\ Fitness\ (h_j)$$

هر چه تناسب فرضیه‌ای بیشتر باشد احتمال انتخاب آن بیشتر است. این احتمال همچنین با مقدار تناسب فرضیه‌های دیگر نسبت عکس دارد.

2. *Crossover*: با استفاده از احتمال بدست آمده توسط رابطه فوق، تعداد $(rp)/2$زوج فرضیه از میان P انتخاب و با استفاده از اپراتور *Crossover*دو فرزند از آنان ایجاد کنید .فرزندان را به P_s اضافه کنید.
3. *Mutate*: تعداد m درصد از اعضا P_s را با احتمال یکنواخت انتخاب و یک بیت از هر یک آنها را به صورت تصادفی معکوس کنید
4. *Update*: $P \leftarrow P_s$
5. برای هر فرضیه h در P مقدار تابع *Fitness* را محاسبه کنید

نمایش فرضیه‌ها

- در الگوریتم ژنتیک معمولا فرضیه‌ها به صورت رشته‌ای از بیت‌ها نشان داده می‌شوند تا اعمال اپراتورهای ژنتیکی برروی آنها ساده‌تر باشد.
  - *Phenotype*: به مقادیر یا راه‌حلهای واقعی گفته می‌شود.
  - *Genotype*: به مقادیر انکد شده یا کروموزم‌ها گفته می‌شود که مورد استفاده GA قرار می‌گیرند.
  - باید راهی برای تبدیل این دو نحوه نمایش به یکدیگر بدست آورده شود.



مثال: نمایش قوانین If-then rules

- برای نمایش مقادیر یک ویژگی نظیر Outlook که دارای سه مقدار Sunny, Overcast, Rain است می‌توان از رشته‌ای با طول 3 بیت استفاده نمود:

100 -> *Outlook = Sunny*

011-> *Outlook = Overcast* $\vee$ *Rain*

برای نمایش ترکیب ویژگی‌ها رشته بیت‌های هر یک را پشت سر هم قرار می‌دهیم:

Outlook	*Wind*
011	10

به همین ترتیب کل یک قانون if- then را می‌توان با پشت سر هم قرار دادن بیت‌های قسمت‌های شرط و نتیجه ایجاد نمود:

IF *Wind = Strong* THEN *PlayTennis = No*

Outlook	*Wind*	*PlayTennis*
111	10	0

$\Rightarrow$ bit string: 111100

نمایش فرضیه‌ها: ملاحظات

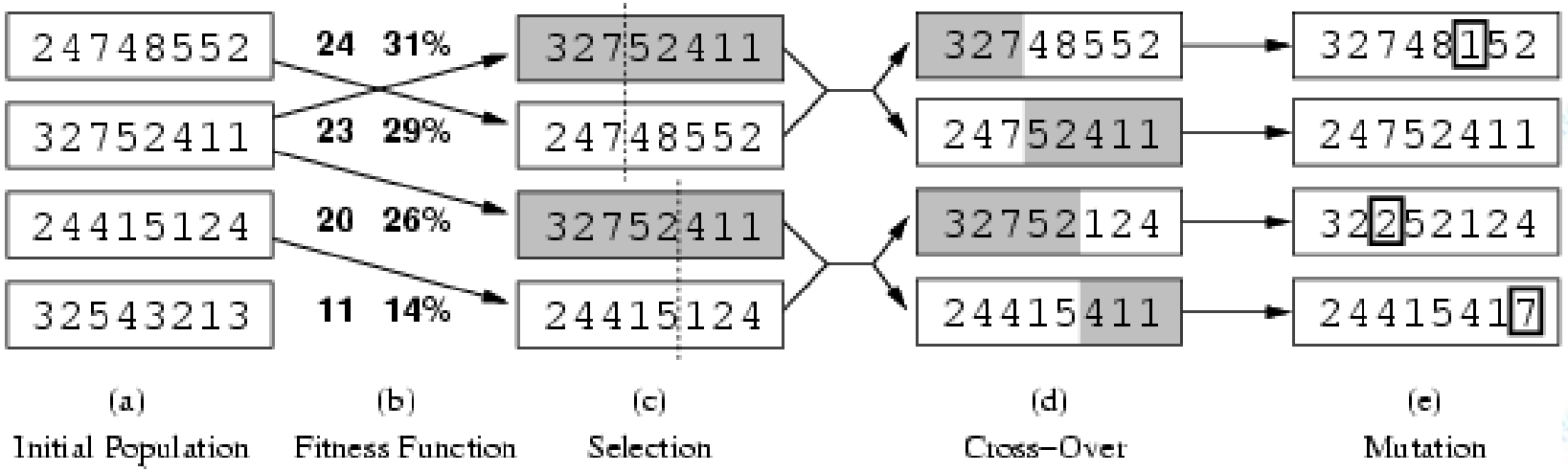
- ممکن است ترکیب بعضی از بیت‌ها منجر به فرضیه‌های بی‌معنی گردد. برای پرهیز از چنین وضعیتی:

  – می‌توان از روش انکدینگ دیگری استفاده نمود.

  – می‌توان اپراتورهای ژنتیکی را طوری تعیین نمود که چنین حالتهائی را حذف نمایند.

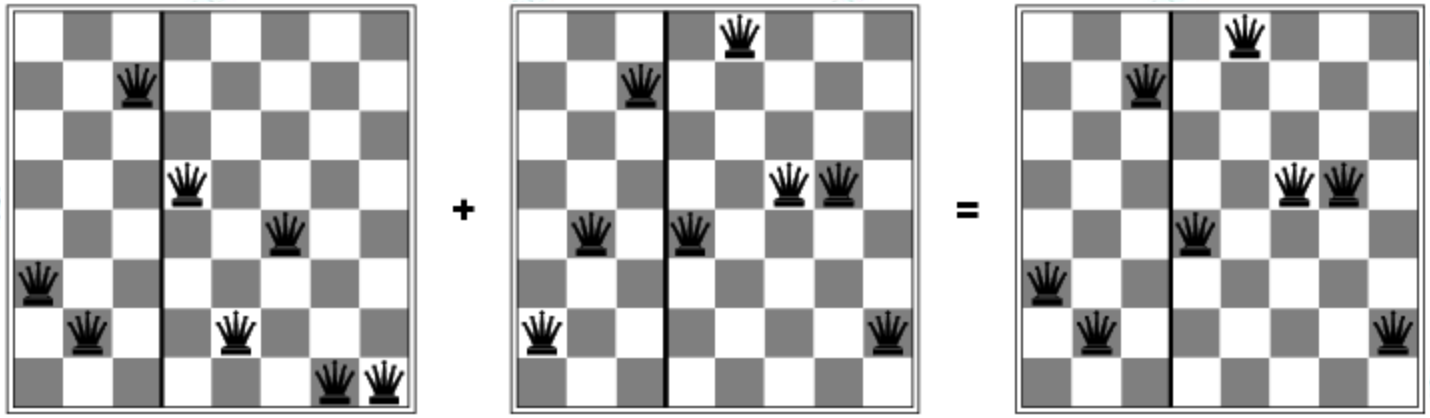
  – می‌توان به این فرضیه‌ها مقدار fitness خیلی کمی نسبت داد.

مثال: مساله ۸-وزیر



- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)
-
- 24/(24+23+20+11) = 31%
-
- 23/(24+23+20+11) = 29% etc
-

مثال: مساله ۸-وزیر



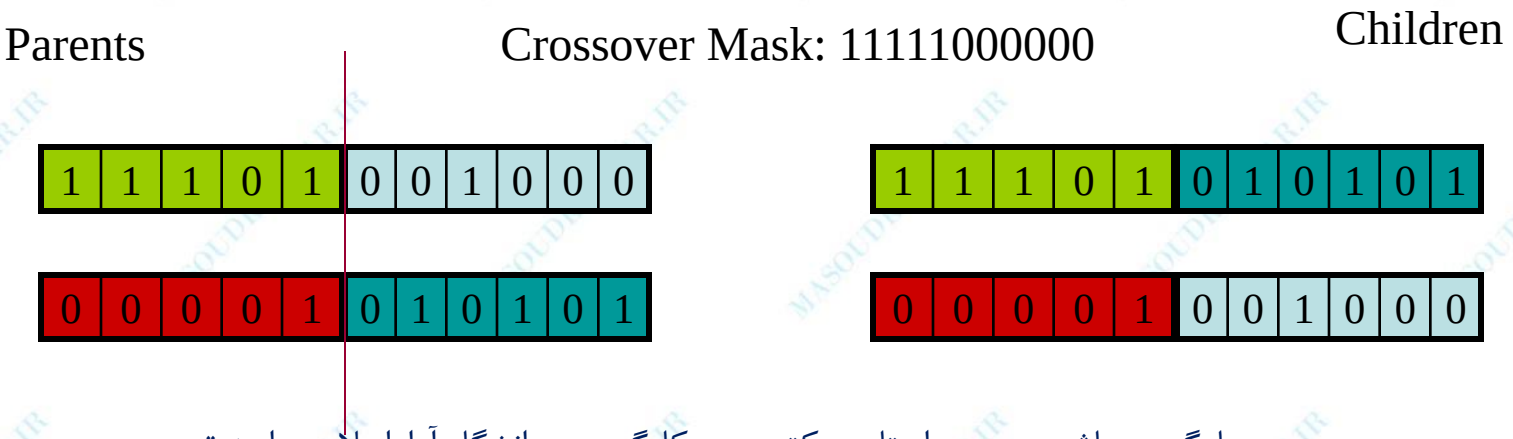
اپراتورهای ژنتیکی: Crossover

- اپراتور Crossover با استفاده از دو رشته والد دو رشته فرزند بوجود می‌آورد.

- برای این کار قسمتی از بیتهای والدین در بیتهای فرزندان کپی می‌شود.
- انتخاب بیت‌هائی که باید از هر یک از والدین کپی شوند به روشهای مختلف انجام می‌شود:
  - single-point crossover
  - Two-point crossover
  - Uniform crossover
- برای تعیین محل بیتهای کپی شونده از یک رشته به نام Crossover Mask استفاده می‌شود.

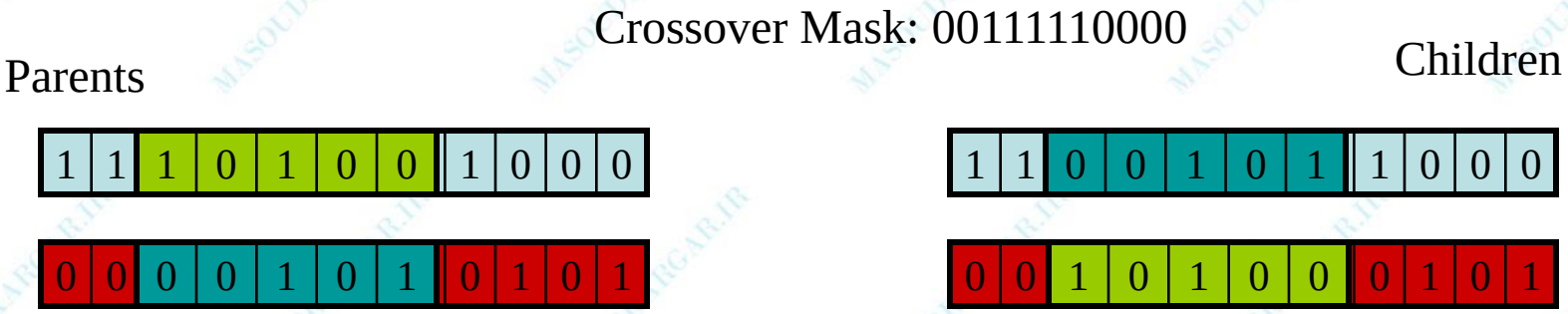
Single-point crossover

- یک نقطه تصادفی در طول رشته انتخاب می‌شود.
- والدین در این نقطه به دو قسمت شکسته می‌شوند.
- هر فرزند با انتخاب تکه اول از یکی از والدین و تکه دوم از والد دیگر بوجود می‌آید.



دیگر روشهای Crossover

- Two-point crossover



- Uniform crossover

بیتها به صورت یکنواخت از والدین انتخاب می‌شوند

اپراتورهای ژنتیکی: Mutation

- اپراتور *mutation* برای بوجود آوردن فرزند فقط از یک والد استفاده می‌کند. این کار با انجام تغییرات کوچکی در رشته اولیه به وقوع می‌پیوندد.
- با استفاده از یک توزیع یکنواخت یک بیت به صورت تصادفی انتخاب و مقدار آن تغییر پیدا می‌کند.
- معمولا *mutation* بعد از انجام *crossover* اعمال می‌شود.

Crossover OR mutation?

- این سوال‌ها سالها مطرح بوده است:

کدامیک بهتر است؟ کدامیک لازم است؟ کدامیک اصلی است؟

- پاسخی که تاکنون بیشتر از بقیه پاسخها مورد قبول بوده:
  - بستگی به صورت مسئله دارد.
  - در حالت کلی بهتر است از هر دو استفاده شود.
  - هر کدام نقش مخصوص خود را دارد.
  - می‌توان الگوریتمی داشت که فقط از *mutation* استفاده کند ولی الگوریتمی که فقط *از crossover* استفاده کند کار نخواهد کرد.

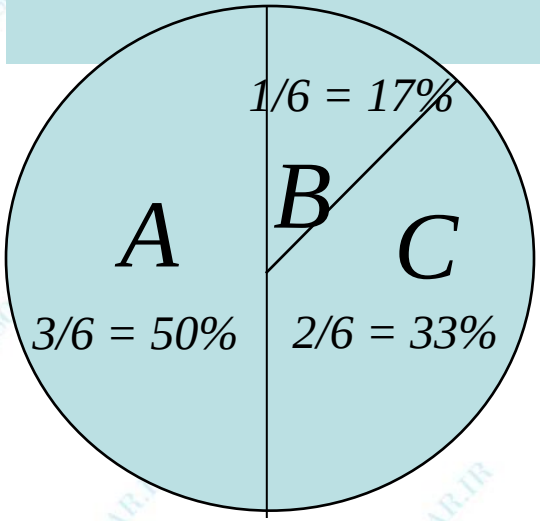
Crossover OR Mutation?

- *Crossover* خاصیت جستجوگرانه و یا *explorative* دارد. می‌تواند با انجام پرشهای بزرگ به محل‌هائی در بین والدین رفته و نواحی جدیدی را کشف نماید.
- *Mutation* خاصیت گسترشی و یا *exploitive* دارد. می‌تواند با انجام تغییرات کوچک تصادفی به نواحی کشف شده وسعت ببخشد.
- *Crossover* *اطلاعات* والدین را ترکیب می‌کند در حالیکه *mutation* می‌تواند اطلاعات جدیدی اضافه نماید.
- برای رسیدن به یک پاسخ بهینه یک خوش شانسی در *Mutation* لازم است.

تابع تناسب

- تابع *fitness* معیاری برای رتبه‌بندی فرضیه‌هاست که کمک می‌کند تا فرضیه‌های برتر برای نسل بعدی جمعیت انتخاب شوند. نحوه انتخاب این تابع بسته به کاربرد مورد نظر دارد:
- *Classification*: در این نوع مسایل تابع تناسب معمولا برابر است با دقت قانون در دسته‌بندی مثالهای آموزشی.

انتخاب فرضیه‌ها



$fitness(A) = 3$

$fitness(B) = 1$

$fitness(C) = 2$

- *Roulette Wheel selection*

در روش معرفی شده در الگوریتم ساده GA احتمال انتخاب یک فرضیه برای استفاده در جمعیت بعدی بستگی به نسبت $fitness$ آن به $fitness$ بقیه اعضا دارد.

این روش *Roulette Wheel selection* نامیده می‌شود.

$$P(h_i) = Fitness\ (h_i) / \Sigma_j\ Fitness\ (hj)$$

- روشهای دیگر:
  - *Tournament selection*
  - *Rank selection*

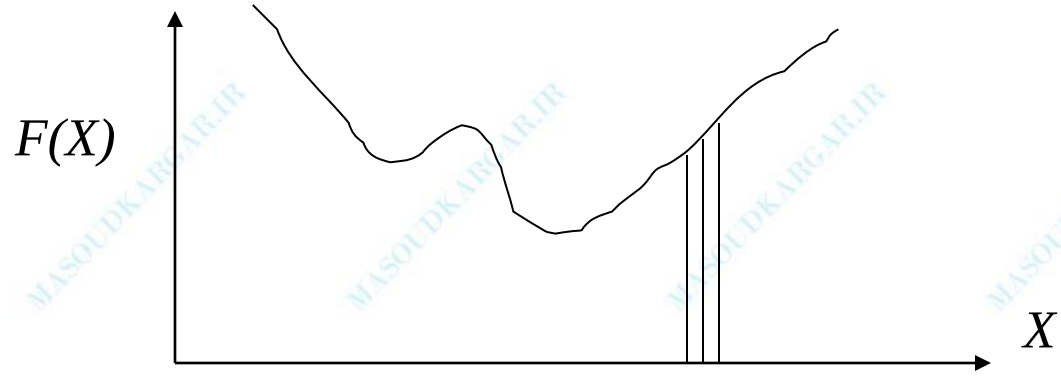
نحوه جستجو در فضای فرضیه

- روش جستجوی *GA* با روشهای دیگر مثل شبکه‌های عصبی تفاوت دارد:
- در شبکه عصبی روش *Gradient descent* به صورت هموار از فرضیه‌ای به فرضیه مشابه دیگری حرکت می‌کند، در حالیکه *GA* ممکن است به صورت ناگهانی فرضیه والد را با فرزندی جایگزین نماید که تفاوت اساسی با والد آن داشته باشد. از این رو احتمال گیر افتادن *GA*در مینیمم محلی کاهش می‌یابد.
- با این وجود *GA* با مشکل دیگری روبروست که *Crowding* نامیده می‌شود.

Crowding

- *Crowding* پدیده‌ای است که در آن عضوی که سازگاری بسیار بیشتری از بقیه افراد جمعیت دارد به طور مرتب تولید نسل کرده و با تولید اعضای مشابه درصد عمده‌ای از جمعیت را اشغال می‌کند.

- این کار باعث کاهش پراکندگی جمعیت شده و سرعت *GA* را کم می‌کند.



راه‌حل رفع مشکل Crowding

- استفاده از *Ranking* برای انتخاب نمونه‌ها: با اختصاص رتبه به فرضیه‌ها از مقدار مطلق تابع تناسب صرف نظر می‌شود، که این باعث می‌شود نسبت به رولت ویل سهم فرضیه‌های با تطابق بالاتر کاهش یابد.
- *Fitness sharing*: مقدار *Fitness* یک عضو در صورتیکه اعضا مشابهی در جمعیت وجود داشته باشند، کاهش می‌یابد.
- محدود کردن اعضائی که می‌توانند با هم ترکیب شوند: برای مثال اعضا به صورت مکانی توزیع شده و فقط به اعضا شبیه به هم امکان تولید نسل داده می‌شود. این کار به ایجاد گروههائی از اعضا مشابه در داخل جمعیت منجر خواهد شد.

چرا GA کار می‌کند؟

- سئوالی که ممکن است برای تازه واردین به روشهای ژنتیکی ایجاد شود این است که آیا این روش واقعا می‌تواند کار مفیدی انجام دهد؟

ارزیابی جمعیت و قضیه Schema

- آیا می‌توان تکامل در جمعیت در طی زمان را به صورت ریاضی مدل نمود؟
- قضیه *Schema* می‌تواند مشخصه پدیده تکامل در *GA* را بیان نماید.
- یک *Schema* مجموعه‌ای از رشته بیت‌ها را توصیف می‌کند. یک *Schema* هر رشته‌ای از *0,1,** است. مثل *0*10* که * حالت *dont care* است.
- یک رشته بیت را می‌توان نماینده هر یک از *Schema* های متفاوتی دانست که با آن تطابق دارند. مثلا *0010* را می‌توان نماینده 2^4 *Schema* مختلف دانست : *0*10,****,00**** و غیره.

قضیه Schema

- قضیه *Schema* بیان می‌کند که چگونه یک *Schema* در طول زمان در جمعیت تکامل پیدا خواهد کرد.
- فرض کنید که در لحظه t تعداد نمونه‌هائی که نماینده یک *Schema* مثل s هستند برابر با $m(s,t)$ باشد. این قضیه مقدار مورد انتظار $m(s,t+1)$ را مشخص می‌کند.
- قبلا دیدیم که احتمال انتخاب یک فرضیه برابر بود با:

$$P(h_i) = Fitness\ (h_i) / \Sigma_j\ Fitness\ (h_j)$$

- این مقدار احتمال را می‌توان به صورت زیر نیز نشان داد:

$$P(h_i) = f\ (h_i) / n\ \bar{f}\ (t)$$

مقدار متوسط *fitness* برای تمامی فرضیه‌های موجود در جمعیت

قضیه Schema

- اگر عضوی از جمعیت انتخاب شود احتمال اینکه این عضو نماینده s باشد برابر است با:

$$p(h \in s) = \sum_{h \in s \cap P_s} \frac{f(h)}{n\,\bar{f}(t)} = \frac{u(s,t)}{n\,\bar{f}(t)} m(s,t)$$

- که در آن مقدار $u(s,t)$ برابر است با مقدار میانگین *fitness* اعضای s:

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

- از این رو مقدار مورد انتظار برای نمونه‌هائی از s که از n مرحله انتخاب مستقل حاصل خواهند شد برابر است با:

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

قضیه Schema

- رابطه فوق به این معناست که تعداد *Schema* های مورد انتظار در لحظه $t+1$ متناسب با مقدار میانگین $u(s,t)$ بوده و با مقدار *fitness* اعضا جمعیت در لحظه نسبت t عکس دارد.
- برای بدست آوردن رابطه فوق فقط اثر مرحله انتخاب نمونه‌ها در نظر گرفته شده است. با در نظر گرفتن اثر *Crossover* و *Mutation* به رابطه زیر خواهیم رسید:

$$E[m(s,t+1)] \geq \frac{u(s,t)}{\bar{f}(t)} m(s,t)\left(1-p_c\frac{d(s)}{l-1}\right)(1-p_m)^{o(s)}$$

Schema Theorem

- *Theorem*

$$E[m(s,t+1)] \geq \frac{\hat{u}(s,t)}{\bar{f}(t)} \cdot m(s,t) \cdot \left(1 - p_c \frac{d_s}{l-1}\right) \cdot (1-p_m)^{o(s)}$$

- $m(s, t)$ ≡ *number of instances of schema s in population at time t*
- $\bar{f}(t)$ ≡ *average fitness of population at time t*
- $\hat{u}(s,t)$ ≡ *average fitness of instances of schema s at time t*
- p_c ≡ *probability of single point crossover operator will be applied to an individual*
- p_m ≡ *probability of mutation operator*
- l ≡ *length of individual bit strings*
- $o(s)$ ≡ *number of defined (non "*") bits in s*
- $d(s)$ ≡ *distance between rightmost, leftmost defined bits in s*

- *Intuitive Meaning*
  - *"The expected number of instances of a schema in the population tends toward its relative fitness"*

خلاصه

- یک *Schema* اطلاعات مفید و امیدبخش موجود در جمعیت را کد می‌کند.
- از آنجائیکه همواره رشته‌هائی که سازگارترند شانس بیشتری برای انتخاب شدن دارند، بتدریج مثالهای بیشتری به بهترین *Schema* ها اختصاص می‌یابند.
- عمل *Crossover* باعث قطع رشته‌ها در نقاط تصادفی می‌شود. با این وجود در صورتیکه این کار باعث قطع *Schema* نشده باشد آنرا تغییر نخواهد داد. در حالت کلی *Schema* های با طول کوتاه کمتر تغییر می‌کنند.
- عمل *mutaion* در حالت کلی باعث تغییرات موثر در *Schema* نمی‌گردد.

*Highly-fit, short-defining-length schema (called **building blocks**) are propagated generation to generation by giving exponentially increasing samples to the observed best*

تفاوت GA با سایر روشهای جستجو

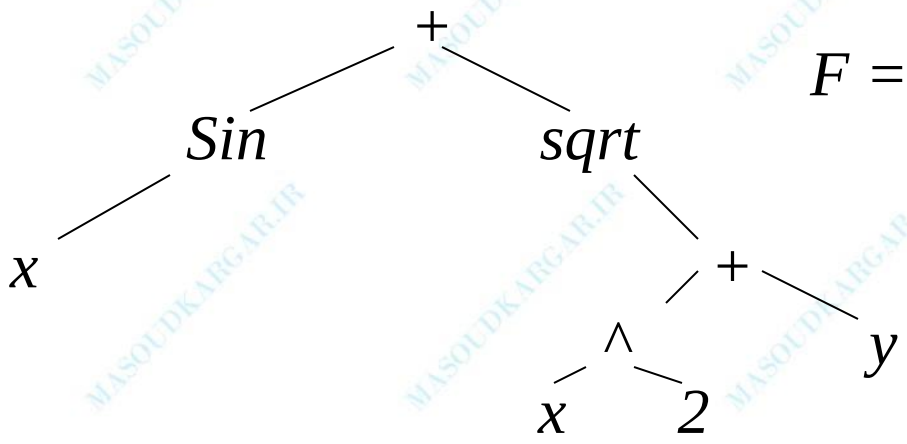
- GA به جای کد کردن پارامترها مجموعه آنها را کد می‌کند.
- GA به جای جستجو برای یک نقطه بدنبال جمعیتی از نقاط می‌گردد.
- GA به جای استفاده از مشتق و یا سایر اطلاعات کمکی مستقیما ازاطلاعات موجود در نتیجه بهره می‌گیرد.
- GA به جای قوانین قطعی از قوانین احتمال برای تغییر استفاده می‌کند.

مثال : کاربرد GA در روباتیک

- *EA* در کاربردهای روباتیک زیادی استفاده شده تا روبات را قادر سازد یاد بگیرد تا چگونه با محدودیت‌هایش کنار بیاید.
- ***RoboCup***
- تیم روبوکاپ *Darwin United* در مسابقات شبیه‌سازی فوتبال با استفاده از *GP توانسته* است به نتایج برتری نسبت به تیمهای دیگر برسد.

Genetic Programming

- *GP* تکنیکی است که کامپیوترها را قادر می‌سازد تا به حل مسائل بپردازند بدون آنکه به طور صریح برای آن برنامه‌ریزی شده باشند.
- *GP* روشی از الگوریتمهای تکاملی است که در آن هر عضو جمعیت یک برنامه کامپیوتری است.
- برنامه‌ها اغلب توسط یک درخت نمایش داده شده و اجرای برنامه برابر است با *parse* کردن درخت.



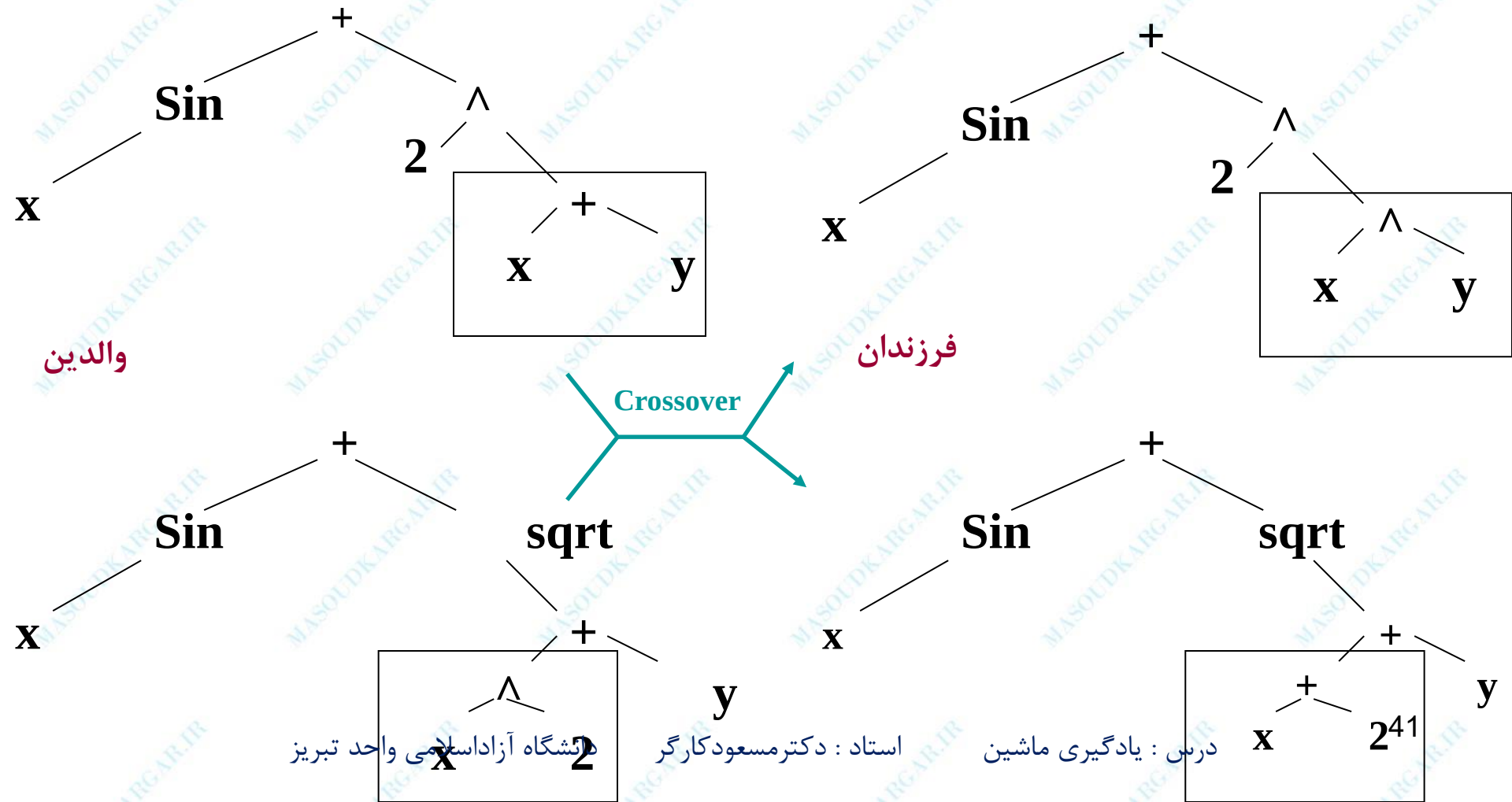
$$F = sin(x) + sqrt(x^2 + y)$$

نمایش برنامه‌ها

- برای استفاده از GP در یک زمینه خاص،
  - می‌بایست توابع پایه‌ای که در آن زمینه مورد نیاز هستند نظیر $sin, cos, sqrt, +, -, etc$ توسط کاربر تعریف شوند.
  - همچنین ترمینالها نظیر متغیرها و ثوابت نیز باید مشخص شوند.
- آنگاه الگوریتم GP در فضای بسیار بزرگ برنامه‌هائی که توسط این مقادیر اولیه قابل بیان هستند، یک عمل جستجوی تکاملی را انجام خوهد داد.

اپراتور crossover برای GP

- اپراتور ***crossover***: شاخه‌هائی از یک درخت پدر با شاخه‌هائی از درخت پدر دیگر به طور تصادفی عوض می‌شوند.



مثال

- در کتاب مثالی از *Koza* مطرح شده که در آن الگوریتمی یاد گرفته می‌شود که بتواند بلوک‌ها را در یک ستون روی هم بچیند.
- هدف مسئله این است که بلوک‌ها طوری روی هم چیده شوند که کلمه *universal* را بسازند.

مثال

- محدودیت: در هر مرحله فقط می‌توان یک بلوک را جابجا نمود. در نتیجه تنها حرکت‌های ممکن عبارتند از:
  - بلوک آخر ستون را می‌توان روی میز قرار داد و یا اینکه
  - یک بلوک را از میز به انتهای ستون منتقل نمود.
- توابع اولیه:
  - *CS (current stack)*: نام بلوک موجود در انتهای ستون را بر می‌گرداند (F برای حالتی که بلوکی وجود ندارد)
  - *TP (top correct block)*: نام آخرین بلوکی را که همراه با بلوک‌های زیرینش ترتیب صحیح مورد نظر را دارند، برمی‌گرداند.
  - *NN (next necessary)*: نام بلوکی که باید در بالای TP قرار گیرد تا ترتیب *universal* درست در بیاید.

مثال

- سایر توابع اولیه:

- *(MS x) move block x to stack* اگر بلوک x روی میز باشد این اپراتور آنـرا بـه بالای ستون منتقل می‌کند. در غیر این صورت مقدار F را برمی‌گرداند.
- *(MT x) move block x to table* اگر بلوک x جائی روی ستون باشد این اپراتور بلوک بالای ستون را به میز منتقل می‌کند. درغیر این صورت مقدار F را برمی‌گرداند.
- *EQ(x y) returns true if x = y.*
- *(NOT x) returns the complement of x.*
- *DU(x y) do x until expression y is true*

مثال

مقدار تابع *fitness*

- در این آزمایش تعداد *166* مثال که هر یک در برگیرنده آرایش اولیه متفاوتی برای بلوک‌ها بودند تدارک دیده شده بود.
- تابع *fitness* یک برنامه برابر است با تعداد مثالهائی که برنامه قادر به حل آن است.
- برنامه با جمعیت اولیه‌ای برابر با *300* برنامه تصادفی شروع بکار نموده و پس از تولید *10* نسل قادر می‌شود تا برنامه‌ای پیدا نماید که تمامی *166* مثال را حل نماید:

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

مثال : طراحی فیلتر

صورت مسئله: برنامه‌ای که یک مدار ساده اولیه را به مدار پیچیده مورد نیاز تبدیل نماید.

- توابع اولیه:
  - تابعی برای اضافه کردن قطعات و سیم‌بندی‌های مدار.
  - تابعی برای حذف کردن قطعات و سیم‌بندی‌های مدار.
- تابع *fitness* شبیه‌سازی مدار بدست آمده توسط نرم‌افزار *SPICE* برای مشخص نمودن میزان تطبیق آن با طرح مورد نظر. خطای مدار برای *101* فرکانس مختلف مورد بررسی قرار می‌گرفت.
- جمعیت اولیه: *640000*
- نرخ *crossover*: *89* درصد.
- نرخ *mutation*: *1* درصد.

مثال: طراحی فیلتر

- سیستم بر روی یک کامپیوتر موازی با 64 گره مورد آزمایش قرار گرفت.
- برای نسل اولیه‌ای که به صورت تصادفی ایجاد شدند در 98% مواقع حتی امکان شبیه‌سازی وجود نداشت.
- این نرخ به تدریج کاهش یافته و پس از تولید 137 نسل مداری حاصل شد که با مشخصات مورد نظر صدق می‌کرد.

در اغلب موارد کارائی الگوریتم GP بستگی به نحوه نمایش و همچنین تابع fitness دارد.

مدلهای تکامل

- در سیستمهای طبیعی هر موجود زنده در طول زندگی خود یاد می‌گیرد که با شرائط سازگاری نماید. به همین ترتیب نسل‌های مختلف یک نمونه در طول زمان سازگاری‌های مختلفی را کسب می‌کنند:
- سوال:

رابطه بین یادگیری یک موجود در طول زندگی شخصی و یادگیری نسل‌های یک نمونه در طول زمان چیست؟

Lamarckian evolution

- *Lamarck* دانشمند قرن نوزدهم فرضیه‌ای ارائه کرده که طبق آن تجربیات یک موجود زنده در ترکیب ژنتیکی فرزندان آن تاثیر می‌گذارد.
- برای مثال موجودی که یاد گرفته از غذای سمی پرهیز کند این ویژگی را به صورت ژنتیکی به فرزندان خود منتقل می‌نماید تا آنها دیگر مجبور به یادگیری این پدیده نباشند.
- اما شواهد تجربی این نظر را تائید نمی‌کنند:

یعنی تجربیات فردی هیچ تاثیری در ترکیب ژنتیکی فرزندان ندارد.

Baldwin Effect

نظریه دیگری وجود دارد که تاثیر یادگیری را بر تکامل توضیح می‌دهد. این نظریه که اثر *Baldwin* نامیده می‌شود بر مبنای مشاهدات زیر استوار است:

- اگر موجودی از طرف محیط متغیری تحت فشار قرار گرفته باشد، افرادی که توانائی یادگیری نحوه برخورد با شرایط را داشته باشند شانس بیشتری برای بقا دارند.

- موجوداتی که تحت شرایط جدید باقی می‌مانند جمعیتی با توانائی یادگیری را تشکیل می‌دهند که فرایندهای تکاملی در آنها سریعتر رخ می‌دهد و باعث می‌شود تا نسلی بوجود بیاید که نیازی به یادگیری مواجهه با شرایط جدید را نداشته باشند.

اجرای موازی الگوریتم‌های ژنتیک

- الگوریتم‌های ژنتیک از قابلیت خوبی برای پیاده‌سازی به صورت موازی برخوردار هستند.
- در یک روش پیاده‌سازی موازی، جمعیت به گروههای کوچکتری با نام *deme* تقسیم شده و هرکدام در یک گره محاسباتی مورد پردازش قرار می‌گیرند.
- در هر گره یک الگوریتم استاندارد *GA* بر روی *deme* اجرا می‌شود.
- انتقال بین گره‌ها از طریق پدیده *migration* صورت می‌پذیرد.

Evolving Neural Networks

از GA برای تکامل جنبه‌های مختلف NN استفاده زیادی بعمل آمده است. از جمله: وزنها، ساختار و تابع یادگیری.

- استفاده از GA برای یادگیری وزنهای یک شبکه عصبی می‌تواند بسیار سریعتر از روش استاندارد *back propagation* عمل نماید.

- استفاده از GA برای یادگیری ساختار شبکه عصبی مشکلتر می‌باشد. برای شبکه‌های کوچک با استفاده از یک ماتریس مشخص می‌شود که چه نرونی به چه نرونهای دیگری متصل است. آنگاه این ماتریس به ژن‌های الگوریتم ژنتیک تبدیل و ترکیبات مختلف آن بررسی می‌گردد.

- برای بدست آوردن تابع یادگیری یک شبکه عصبی راه‌حلهائی نظیر استفاده از GP مورد استفاده قرار گرفته اما عموما این روشها بسیار کند عمل کرده‌اند.