الگوریتم ژنتیک و حل مساله TSP

پیام خان تیموری
payam_csut@yahoo.com

چکیده مقاله : در این مقاله ابتدا الگوریتمهای ژنتیک را معرفی کرده و مراحل انجام چنین الگوریتمهایی توضیح داده می شود. بعد از اینکه یک دید کلی نسبت به الگوریتمهای ژنتیک پیدا کردیم به مساله Traveling Salesman Problem می پردازیم. ابتدا چند روشی که برای حل TSP ارائه شده است را بیان می کنیم و بعد سعی می کنیم الگوریتمهای ژنتیک مختلفی را برای این مساله مطرح کنیم و سپس بررسی می کنیم که کدام یک از این الگوریتمهای ژنتیک بهتر از بقیه روشها جواب می دهند. در پایان نیز مقایسه ای بین الگوریتمهای ژنتیک و دیگر الگوریتمها انجام می دهیم.

کلمات کلیدی:

Gene – Chromosome – Allele – Encoding – Population – Evaluation – Fitness – Selection – Crossover – Mutation – Decoding – Generation – Parent

مقدمه:

الگوریتمهای ژنتیك ابزاری می باشند که توسط آن ماشین می تواند مکانیزم انتخاب طبیعی را شبیه سازی نماید. این عمل با جستجو در فضای مسئله جهت یافتن جواب برتر و نه الزاما بهینه صورت می پذیرد.

الگوریتم های ژنتیك با توجه به نظریه داروین در مورد تکامل ، جان گرفتند . سپس نظریه محاسبات تکاملی ، توسط ریچنبرگ در سال 1960 معرفی شدند و این نظریه توسط محققان دیگر توسعه یافت تا در سال 1975 منجر به اختراع الگوریتم های ژنتیك توسط هالاند Holland و دانشجویانش شد.

در الگوریتم های ژنتیک یک سری تعاریف اولیه داریم که در زیر آمده است:

1. ژن (gene) : واحد پایه ژنتیک است.
2. فرم (allele) : حالتهای مختلف هر ژن را می گویند.
3. کروموزوم (choromosome) : به گروهی از ژنها اطلاق می شود.

بعد از تعاریف بالا مفاهیمی از قبیل Encoding , Evaluation , Crossover , Mutation مطرح می شود که بر اساس سه تعریف بالا مطرح می شوند.

اصول الگوریتم ژنتیک :

حال به شرح مراحل ذکر شده درمقدمه می پردازیم.

1. *Encoding :*

این مرحله شاید مشکلترین مرحله حل مساله به روش الگوریتم ژنتیک باشد.این مرحله به این مفهوم است که ما باید با ارائه یک شبیه سازی و جایگذاری (Representation) خوب برای کلیه جوابهای ممکن مراحل بعدی را ادامه دهیم. اهمیت این مرحله به این دلیل است که نحوه ادامه کار به این مرحله بستگی دارد.

در واقع ما در این مرحله رشته های کروموزومی یا همان رشته های بیتی ممکن برای جوابها را می سازیم. چه بسا ما بتوانیم با یک شبیه سازی خوب برای جوابها الگوریتم را در یک زمان خوب و معقول پیش ببریم. بعد از ساختار بندی برای هر جواب ممکن از کنار هم گذاشتن این ساختارها جمعیت اولیه ما (First Population) ساخته می شود.

برای مثال برای شبیه سازی اعداد یک روش معمول که به کار می رود عبارتست از تبدیل عدد به یک رشته باینری (binary).

13 → 1 1 0 1 (رشته با طول 4)
9 → 1 0 0 1

2. *Evaluation* :

این مرحله نیز نقش بسیار مهمی را در حل مساله به روش الگوریتم ژنتیک ایفا می کند. ما در این مرحله با معرفی یک معیار و اندازه به بهتر بودن یا نبودن هر جواب ممکن از جمعیت اولیه پی می بریم.

یعنی اینکه این معیار به ما میگوید که مثلا جواب x واقع در جمعیت اولیه (First Population) چقدر خوب است.

این معیار می تواند با نسبت دادن یک عدد به هر جواب ممکن میزان خوب بودن آن جواب را بیان کند. مثلا اگر داشته باشیم

f(a)=3 و f(b)=5 این به این معنی است که عضو b در جمعیت اولیه بهتر از a است یا مقدار Fitness بالاتری نسبت به a دارد.

3. *Crossover :*

به این مطلب اشاره شد که در الگوریتم ژنتیک برای بوجود آوردن یک نسل جدید(یا یک سری کروموزوم جدید) از جوابها ما یک سری از تغییرات را روی جوابهای انتخاب شده (Selected) یا کروموزومها اعمال می کنیم. به این سری از تغییرات اعمال عملگر می گوییم.

حال عملگرهای ما در این تغییرات دو عملگر Crossover و Mutation می باشد که ما در این قسمت به نحوه عملکرد Crossover می پردازیم.

عملگر Crossover روی جوابهای انتخاب شده برای تغییر به این گونه عمل می کند که یک نقطه به صورت تصادفی روی رشته کروموزومی انتخاب می کند (که البته random است) بعد ناحیه های چپ یا راست آن نقطه در رشته کروموزومی جا به جا میشود.

در (Figure 1) نمونه هایی از crossover با یک نقطه انتخابی را مشاهده می کنید.

همچنین برای crossover می توانیم دو نقطه انتخاب کنیم و ناحیه های بین دو نقطه را جا به جا کنیم. (Figure 2)

4. *Mutation :*

Mutation نیز عملگر دیگری هست که جوابهای ممکن دیگری را متولد می کند.

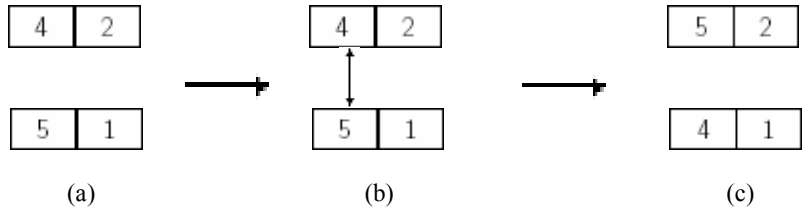
نحوه عملکرد آن به این گونه است به صورت تصادفی روی کروموزوم انتخابی نقطه ای را انتخاب کرده و فرم (allele) آن را تغییر می دهد.(Figure 3)

در شکل یک بعد از انتخاب نقطه فرم آن تغییر می کند و می تواند عدد دیگری مثل 2 اختیار کند (شکل دو). کروموزوم جدید نیز خود تغییر می کند و شکل سه را می سازد.

5. *Decoding :*

عکس عمل encoding است.

در این مرحله بعد از اینکه الگوریتم بهترین جواب را برای مساله ارائه کرد لازم است عکس عمل رمزگذاری روی جوابها یا همان عمل decoding اعمال شود تا بتوانیم نسخه واقعی جواب را به وضوح در دست داشته باشیم.



a) در این مرحله دو کروموزوم یا دو رشته والد انتخاب می شوند. (شیوه های انتخاب گفته خواهد شد.)

b) در این مرحله یک نقطه (point) به صورت تصادفی(random) روی کروموزوم های هم طول انتخاب می شود.

c) در مرحله آخر یک قسمت (چپ یا راست) دو کروموزوم از نقطه انتخاب شده با هم جا به جا می شوند.

Father Chromosome	***Mother Chromosome***	***Crossover Point***	***Child Chromosome***
10010011	10110110	3	100|10110
10000000	10110110	6	100000|10
10110110	11101110	2	10|101110
10110110	11101110	5	10110|110

One-point Crossover Examples

(Figure 1)

Father Chromosome	***Mother Chromosome***	***Cross Point 1***	***Cross Point 2***	***Child Chromosome***
10010011	10110110	3	6	100|1011|1
10000000	10110110	0	4	10000|110
10110110	11101110	2	3	10|10|0110

Two-Point Crossover Examples

(Figure 2)

(Figure 3)

الگوریتم ژنتیک :

الگوریتم ژنتیک با انجام قدم به قدم مراحل زیر پیشرفت می کند:

1. شروع الگوریتم با یک جمعیت متشکل از n فرد تصادفی که هر کدام کروموزومی به طول l دارند.
2. محاسبه fitness برای هر فرد
3. انتخاب دو فرد بر اساس بالاتر بودن fitness (انتخاب والدین)
4. اعمال crossover و تولد بچه ها از والدین
5. اعمال mutation با احتمال p برای هر بیت
6. قرار دادن بچه های متولد شده داخل یک مجموعه به عنوان نسل جدید
7. تغییر دادن جمعیت اولیه همراه ورود نسل جدید (New Generation)
8. رفتن به قدم 2

در ابتدای الگوریتم ژنتیک یک جمعیت اولیه شامل n جواب ممکن تصادفی درست می کنیم. بعد تابع evaluation مقدار fitness هر جواب را حساب می کند. یک تابع selection بر اساس اینکه هر جوابی که fitness بالاتری داشت احتمال انتخاب بیشتری دارد دو parent انتخاب می کند.

برای مرحله بعد عملگرهای crossover و mutation روی کروموزوم های انتخاب شده اعمال می شود.

بعد از این مرحله تعدادی از جوابهای بد یعنی با fitness پایین حذف می شوند و جای خود را به بچه های متولد شده و نسل جدید

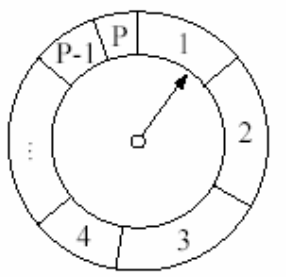
(new generation) می دهند. بعد از تثبیت نسل جدید الگوریتم بار دیگر به مرحله evaluation برمی گردد.

همان طور که اشاره شد عمل selection (مرحله 3) براین اساس انجام می شود که همه جوابها برای parent شدن هم شانس نیستند ; به عبارت دیگر جواب هایی احتمال زیاد و بالاتری برای parent شدن دارند که مقدار fitness آنها بالاتر باشد.

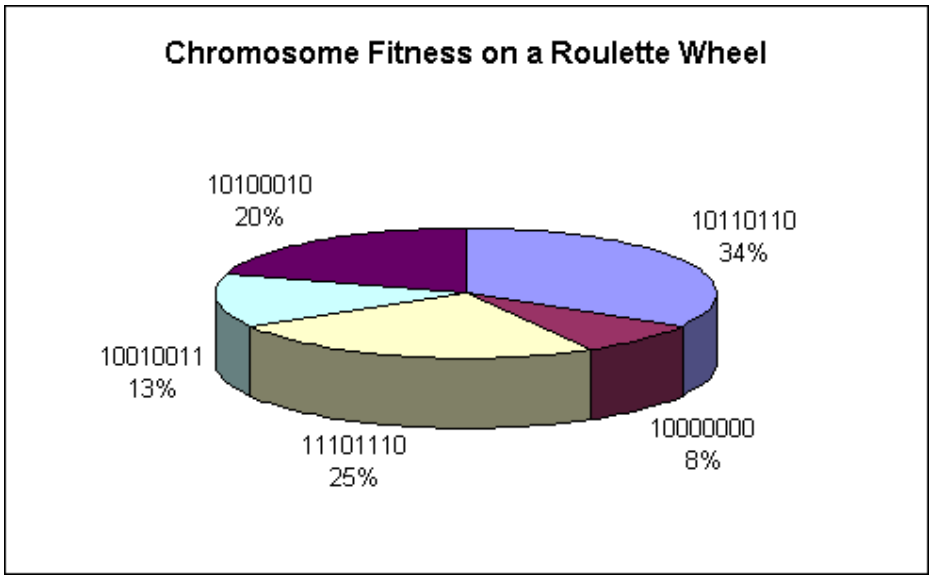
ما می توانیم برای هر جواب یک عدد به عنوان fitness نظیر کنیم , مثلا برای جواب بودن x=2 عدد 4 و جواب بودن x=0 عدد 2 را نسبت دهیم در این صورت مشخص است که x=2 جواب بهتری نسبت به x= 0 است. حال می توانیم تابع احتمال و انتخاب را به این گونه تعریف کنیم : (جمعیت p و اعداد نظیر شده q (quality) فرض شده است.)

$$f_i = \frac{q_i}{\sum_j^P q_j}$$

این نوع selection با عنوان Roulette Wheel Selection بیان می شود که (Figure 4) نشان می دهد.



Chromosome	***Fitness***
10110110	20
10000000	5
11101110	15
10010011	8
10100010	12



Pie Chart of Fitness

(Figure 4)

حال با اشاره کردن به مثالی مراحل انجام الگوریتم را نشان می دهیم :

مثال: فرض کنید تابع $f(x) = -x^2 + 6x - 3$ را داریم . می خواهیم برای $0 \le x \le 15$ و $x \in N$ ماکزیمم تابع را بیابیم.

مراحل حل مساله به روش الگوریتم ژنتیک به شرح زیر است :

1. encoding

برای اینکه ما جوابهای ممکن یعنی $0 \le x \le 15$ را کدگذاری کنیم یکی از روشهایی که در پیش داریم تبدیل هر عدد به یک عدد باینری متناظر است. چون بیشترین جواب ممکن ما 15 است و متناظر باینری آن 1 1 1 1 چهار بیتی است لذا تمام جواب های ما (کروموزوم ها) دارای طولی برابر 4=l می باشند.

2. Evaluation

تابعی که مقداری برای fitness حساب می کند مستقیما از خود تابع f(x) بدست می آید.

یعنی اگر به ازای یک a و b : $f(b) > f(a)$ باشد آنگاه b دارای fitness بیشتری است . یا اینکه b از a بهتر است.

3. Selection

فرض کنیم که در آن جمعیت اولیه با m فرد که $m < n$ دوعدد 3 و8 دارای fitness بیشتری نسبت به جمعیت خود بودند و احتمال انتخاب بیشتری داشتند. بعد از انتخاب این دو عدد به عنوان parent الگوریتم وارد مراحل تولید نسل جدید می شود.

$3 \rightarrow 0\,0\,1\,1 \qquad f(3) = 6$

$8 \rightarrow 1\,0\,0\,0 \qquad f(8) = -19$

4. crossover

فرض کنیم که crossover عمل زیر را انجام دهد.

$$\begin{matrix} 0\,0\,1\,1 \\ 1\,0\,0\,0 \end{matrix} \xrightarrow{\text{point} = 3} \begin{matrix} 0\,0\,1\,0 \\ 1\,0\,0\,1 \end{matrix}$$

5. mutation

و نیز عملگر mutation :

$$\begin{matrix} 0\,0\,1\,0 \\ 1\,0\,0\,1 \end{matrix} \xrightarrow{\text{point} = 2} \begin{matrix} 0\,1\,1\,0 = (6) \\ 1\,1\,0\,1 = (13) \end{matrix}$$

حال کروموزوم های متولد شده جزو نسل جدید به حساب می آیند و به جمعیت اولیه افزوده می شوند وجوابهای با fitness پایین حذف می شوند و الگوریتم دوباره با n فرد به کار خود ادامه می دهد.

یک نکته لازم به تذکر است که ممکن است بهترین جواب (برای مثال $x = 6$) طی مراحل بعد به نحوی از بین برود. برای جلوگیری از این امر بعد از هر بار انجام الگوریتم بهترین جواب را در جایی

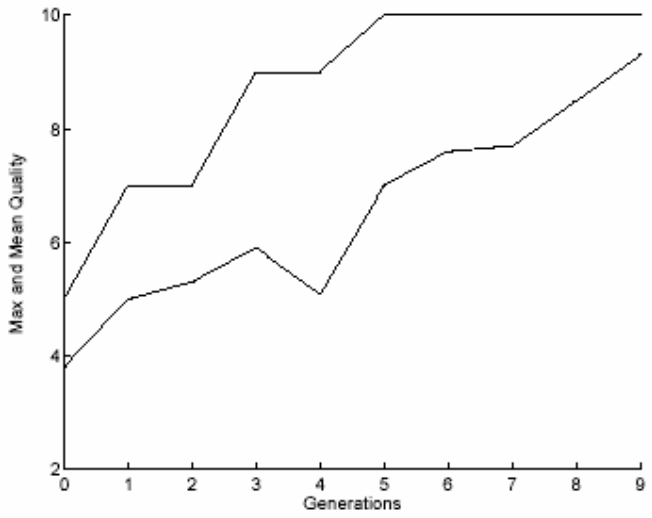
کنار می گذاریم (far) تا همیشه بهترین جواب حاصل از هر بار اجرای الگوریتم را داشته باشیم.

معیارهای مختلفی را می توان برای توقف الگوریتم در نظر گرفت و معمولا چند معیار برای توقف استفاده می شود تا احتمالهای مختلف وقوع پیشامدها در طی اجرای الگوریتم حساب شوند. یک معیار می تواند این باشد که بهترین جواب را بعد از اجرای تعداد مشخصی بار از الگوریتم تغییر ندهد. یا معیار دیگر اینکه میانگین fitness جوابهای موجود در جمعیت جاری همان fitness بهترین جواب یا بسیار نزدیک به آن باشد.

و یا اینکه می توانیم از پیش قرارداد کنیم که الگوریتم به تعداد مشخصی اجرا شود. معمولا روتین های توقف مختلف است و بستگی به پیچیدگی و چگونگی مساله دارد.

در اینجا ما برای تعریف تابع evaluation از خود تابع استفاده کردیم و لزومی برای نسبت دهی یک مقدار به عنوان quality نبود.

اگر ما این مراحل را به گونه ای دنبال کنیم که بعد از هر بار اجرا جواب های بد حذف و جوابهای نسل جدید که احتمالا fitness بالایی دارند جایگزین آنها شوند به مرور فاصله fitness بین min و Max کم می شود و میانگین جواب ها به بهترین جواب میل می کند و این نشان می دهد که ما به طور کلی به بهترین جواب نزدیک می شویم.



سوالی که در اینجا مطرح می شود اینست که اگر در مثال قبل به جای فضای گسسته فضای پیوسته [0 , 15] مطرح می شد الگوریتم به چه شکلی انجام می گرفت؟

در این حالت ما یک ε تعریف می کنیم تا هر بیت جواب در مراحلی مثل mutation به اندازه ε تغییر کند. بدیهی است ما هر چقدر این ε را کوچکتر فرض کنیم میزان محاسبات و تکرار الگوریتم بالا می رود و به همین دلیل زمان اتمام الگوریتم بالا می رود. عامل دیگری که در زمان موثر است n اولیه در جمعیت اولیه است که با زیاد فرض کردن n زمان اتمام بالا می رود.

حال به ذکر دو نکته می پردازیم.

1 . بعد از تکرار الگوریتم به تدریج تفاوت بین fitness جوابها کمتر می شود و در این صورت تابع selection نیز کم کم احتمال یکسانی را برای انتخاب جوابهای ممکن تخصیص می دهد. در این حالت الگوریتم پایان می یابد. (چون بهترین جواب هم احتمال انتخابی برابر دیگر جوابها دارد.)

2 . در جریان الگوریتم ژنتیک احتمال تولید Super - Subject وجود دارد. به این معنی که از یک جواب (کروموزوم) به تعداد بسیار زیادی تولید می شود و این جمعیت را به میزان زیادی انبوه کرده و علاوه بر این باعث می شود که جمعیت ما از تنوع جواب کمتری نسبت به میزان جمعیت برخوردار باشد که این در الگوریتم ایجاد مشکل می کند.

برای جلوگیری از این گونه مشکلات چند ایده مطرح می شود:

(Windowing) : به این مفهوم است که fitness بدترین جواب را از fitness بقیه جوابها کم می کنیم. این عمل موجب می شود که تعدادی جواب با fitness صفر بدست آیند و این یعنی اینکه احتمال انتخابی برابر صفر دارند. و نیز احتمال انتخاب بهترین جواب قوت می گیرد.

(Exponential) : برای fitness هر جواب مقدار جدید ریشه دوم (fitness + 1) را به کار می بریم. این عمل موجب می شود تا اثر قویترین جواب تعدیل شود.

(Linear Transformation) : به کار گرفتن رابطه زیر برای بدست آوردن fitness جدید.

$$f_2 = a (f_1) + b$$

این رابطه هم اثر بهترین جواب را کمتر می کند.

(Linear Normalization) : برای مثال در یک جمعیت شامل 10 شی برای اولین فرد عدد 100 و دومین فرد عدد 90 و ... برای 10 امین فرد عدد 10 را نسبت می دهیم. این عمل موجب می شود که اختلاف بین احتمال انتخاب فقط به تعداد افراد بستگی داشته باشد. در این روش پخش شدن Distribution فقط بر اساس تعداد است نه بر اساس fitness . این روش از پخش شدن Super - Subject جلوگیری می کند.

The Traveling Salesman Problem

مساله فروشنده دوره گرد به این شکل است که می خواهیم در یک تعداد شهر (n) دوری پیدا کنیم که از هر شهر دقیقا یک بار عبور کند و در پایان به شهر آغازین بازگردد به طوری که طول دور مینیمم باشد.

هر چند منشا TSP دقیقا مشخص نیست ولی قطعا در حدود سال 1931 بوده است. اولین نمونه شبیه به این مساله توسط Euler در سال 1759 مطرح شد و به این صورت بود که یک مهره اسب می بایست روی برد شطرنج حرکت کند و از هر خانه دقیقا یک بار عبور کند.

این مساله را می توان به صورت ریاضی هم شبیه سازی کرد. به این ترتیب که ما در یک گراف وزن دار G(v,e) دوری فراگیر (اویلری) با مینیمم مجموع وزنهای یالهای گذرنده می خواهیم بیابیم.

به روش ریاضی مساله با یافتن تعداد جایگشت های n شی متمایز وسپس ارزیابی هر حالت بررسی می شود.

تعداد جایگشتها !n است. برای یافتن مینیمم دورها نیز به حداکثر !n محاسبه احتیاج داریم. ولی اگر n را زیاد فرض کنیم تعداد محاسبات بسیار بالا خواهد بود به همین دلیل گفته می شود که الگوریتم حل مساله در زمان چند جمله ای نیست. (None-Polynomial)

حال می خواهیم چند الگوریتم ارائه شده برای حل مساله TSP را مطرح کنیم. تا به امروز الگوریتمی بدست نیامده است که این مساله را در زمان چندجمله ای (Polynomial) حل کند. به همین دلیل ما بهینگی را فدای زمان می کنیم تا بتوانیم در یک زمان معقول به یک جواب خوب برسیم.

یکی از الگوریتمهای ارائه شده الگوریتم حریصانه است (Greedy). به این شکل که الگوریتم یک لیست از همه یالها در گراف ایجاد می کند و سپس آنها را از کمترین هزینه به بیشترین هزینه مرتب می کند. سپس ابتدا یالی با کمترین هزینه را انتخاب می کند و چک می کند که این یال سبب نشود که دوری در گراف ایجاد شود که از همه رئوس عبور نکند.

روش دیگری نیز هست که شبیه به Greedy است که به نام Nearset Neighbor مطرح شده است.

به این شکل که یک نقطه آغازین به صورت تصادفی انتخاب می کنیم وبعد به نزدیکترین راس بعدی می رویم و از آن راس نیز همین کار را ادامه می دهیم. البته باز هم شرط می کنیم که در هر مرحله دوری که اویلری نباشد ایجاد نشود.

این روشها همیشه راه حل خوبی ارائه نمی دهد چون اغلب آخرین یال اضافه شده وزن نسبتا زیادی دارد.

الگوریتم بعدی A minimum spanning tree میباشد.

ابتدا ما یک درخت پوشای مینیمم با n-1 یال می یابیم. سپس می توانیم یک دور به وسیله رفتار وچگونگی یالها در درخت پوشایمان ایجاد کنیم (مثل یالهای دو جهتی). بعد از یک شهر که فقط به یک شهر دیگر متصل شده است شروع می کنیم و با دنبال کردن یالهای طی نشده برای شهرهای جدید ادامه می دهیم. اگر یال طی نشده ای وجود نداشت به یال قبلی بر می گردیم و این کار را ادامه می دهیم تا به شهر ابتدایی بازگردیم. با این کار ما یک کران بالا برای TSP خواهیم داشت.

قابل توجه است که ما بعضی از شهرها را بیش از یک بار بازدید کردیم. وقتی که نیاز داریم به عقب حرکت می کنیم ولی در عوض به شهر بازدید نشده بعدی می رویم. بعد از اینکه همه شهرها بازدید شدند به شهر شروع برمی گردیم.

بعد از ارائه این روشها به سراغ الگوریتم ژنتیک می رویم.

TSP with genetic algorithm

در این بخش ما مراحلی را که الگوریتم ژنتیک برای TSP می سازد را توضیح می دهیم.

1. Encoding

برای encoding می توانیم یک ماتریس مجاورت گراف ایجاد کنیم که شامل 1 در مکان i و j است اگر یک یال از راس i به راس j وجود داشته باشد و در غیر این صورت 0 است.

حال می توانیم از این ماتریس همان گونه که هست استفاده کنیم یا به این صورت که سطرهای ماتریس را به هم الحاق کنیم و رشته ای طولانی از 0 و 1 ها ایجاد کنیم ولی توجه داریم که این رشته نمایش باینری عدد نیست.

به این صورت اولین متد encoding ساخته می شود.

برای مثال ماتریس مقابل :

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

دوری را نشان می دهد که از شهر1 به شهر 3 و از شهر 3 به شهر 2 و از شهر 2 به شهر 1 می رود.

متد بعدی که مورد بررسی قرار می گیرد متدی است که یک رشته از اعداد که معرف شماره شهر است را می سازد.

اولین شیوه ساخت این متد به این صورت است که برای مثال رشته $v = 1234$ این مفهوم را می رساند که ابتدا از شهر 1 شروع کرده سپس به شهر 2 می رویم و از آنجا به شهر 3 و بعد به شهر 4 حرکت می کنیم. در پایان هم به شهر اول بر می گردیم. توجه کنید که دو رشته 1234 و 2341 معادلند.

شیوه دوم این است که اگر رشته $v = 3124$ را داشته باشیم به این معنی است که دور از شهر 1 به شهر 3 و از شهر 3 به شهر 2 و از شهر 2 به شهر 4 و در پایان از شهر 4 به شهر 2 می رویم.

قابل توجه است که هر رشته ممکن در اینجا یک دور مجاز را نشان نمی دهد. مثلا رشته 3412 نشان می دهد که ما از شهر 1 به شهر 3 می رویم و به شهر 1 بازمی گردیم واز شهر 2 به شهر 4 می رویم و به شهر 2 باز می گردیم که یک رشته بی ربط است.

2. Crossover

متد اول Partially Matched Crossover (PMX) می باشد.

اگر دو رشته مقابل را داشته باشیم :

1 2 3 4 | 5 6 7 | 8
8 5 2 1 | 5 6 7 | 7

و یک crossover دو نقطه ای انجام دهیم خواهیم داشت :

v_1 = 1 2 3 4 | 3 6 4 | 8
v_2 = 8 5 2 1 | 5 6 7 | 7

که به طور بدیهی غیر مجاز هستند چون v_1 شهر 5 یا 7 را بازدید نمی کند و شهر های 3 و 4 را دو بار بازدید می کند. به طور مشابه v_2 شهر های 3 و 4 را نمی بیند و شهر های 5 و 7 را دو بار می بیند.

PMX بدون استفاده از ابزاری این مشکل را به این گونه حل می کند که تعویض های 5 ↔ 3 و 6 ↔ 6 و 7 ↔ 4 را انجام می دهد سپس این تعویض ها را عینا روی ژن های خارج از نقاط crossover تکرار می کند.

و به این ترتیب رشته های زیر ساخته می شود :

1 2 5 7 3 6 4 8
8 3 2 1 5 6 7 4

ولی ما در این روش لزوما یک دور مجاز تولید نمی کنیم به همین دلیل نیاز داریم تا با یک روتین بهتر برای crossover راه حل بهتری پیدا کنیم و دورهایی که مجازند را تولید کنیم.

متد دوم Cycle Crossover (CX) می باشد.

لازم به ذکر است که این متد crossover روی اولین شیوه encoding عمل می کند یعنی رشته 1234 به این مفهوم است که به ترتیب از 1 به 2 و به 3 و به 4 می رویم و در آخر به 1 بر می گردیم.

فرض کنید دو رشته زیر را داریم:

1 2 3 4 5 6 7 8
8 5 2 1 3 6 4 7

ما برای تولید یک رشته جدید v_1 به این شکل عمل می کنیم :

اولین ژن را از یکی از parent ها انتخاب می کنیم:

v_1 = 1 - - - - - - -

باید هر عنصر را از یکی از parent ها برداریم و آنرا در موقعیتی که قبلا بوده قرار دهیم.

از آنجاییکه اولین موقعیت توسط 1 اشغال شده است عدد 8 از رشته دوم نمی تواند به آن محل برود پس مجبوریم 8 را هم از رشته اول برداریم:

v_1 = 1 - - - - - - 8

و به همین شکل 7 و 4 را نیز از رشته اول برداشته و در محل خود قرار دهیم :

v_1 = 1 - 4 - - - 7 8

حال با پر کردن بقیه محل ها با عناصر آن محلها از رشته دوم v_1 را می سازیم.

این متد همیشه یک کروموزوم مجاز می سازد.

البته ممکن است فرزند درست شده همان parent باشد ولی این مشکل نیست چون نشانگر اینست که fitness, parent بالایی دارد و باز هم می تواند یک انتخاب باشد.

متد سوم Order Crossover می باشد که خیلی به PMX شبیه است.

همان دو نقطه را در نظر می گیرد ولی به جای اصلاح کروموزومها با تعویض تکرارها به طور ساده تری بقیه ژن ها را مرتب می کند تا یک دور مجاز بدهد.

مثلا اگر دو رشته زیر را داشته باشیم :

1 3 5 | 7 6 2 | 4 8
5 6 3 | 8 2 1 | 4 7

با تعویض کردن ژنهای بین دو نقطه بدست می آید :

v_1 = - - - | 8 2 1| - -
v_2 = - - - | 7 6 2| - -

سپس با شروع از دومین نقطه crossover ژنها را از کروموزوم parent ثبت می کنیم.

4 8 1 3 5 7 6 2
4 7 5 6 3 8 2 1

سپس ژن هایی را که بین نقاط crossover بوده اند را حذف می کنیم. یعنی 8 و 2 و 1 را از لیست v_1 و 7 و 6 و 2 را از لیست v_2 حذف می کنیم تا رشته های زیر بدست آیند :

4 3 5 7 6
4 5 3 8 1

و با جایگزینی از دومین نقطه crossover با کروموزومهای فرزند رشته های نهایی زیر بدست می آیند:

v_1 = 5 7 6 8 2 1 4 3
v_2 = 3 8 1 7 6 2 4 5

متد چهارم Crossover Metrix می باشد.

همان crossover یک یا دو نقطه ای می باشد.

اگر ماتریس های زیر را داشته باشیم :

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

نقاط crossover را بعد از اولین ستون و بعد از دومین ستون انتخاب می کنیم. crossover کردن ستون ها نتیجه می دهد:

$$A' = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$

$$B' = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

حالا ما چندین 1 در بعضی سطرها داریم و بعضی از سطرها اصلا 1 ندارند. این را بوسیله انتقال یکی از 1 ها از سطری که چندین 1 دارد به سطری که هیچ یکی ندارد درست می کنیم.

این انتخاب بین سطرهای شامل چند 1 تصادفی است.

$$A'' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$B'' = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

حال با ملاحظه ماتریس اول می بینیم که $a \leftrightarrow a$ و $b \rightarrow c \rightarrow b$. پس ما دو دور مختلف ایجاد کرده ایم ولی این را بوسیله برش و متصل کردن درست می کنیم.

یال a به a را و همچنین یکی از یالهای بین b و c را میبریم و a را به b و c وصل می کنیم.

متد پنجم Modified Order Crossover (MOX) شبیه crossover یک نقطه ای است.

یک نقطه crossover تصادفی در parent انتخاب می کنیم و به طور معمول ژنهای قبل از نقطه را همان طور که هستند رها می کنیم. سپس ژنهای بعد از نقطه crossover را به ترتیبی که در دومین کروموزوم parent ظاهر شده اند دوباره مرتب می کنیم.

اکر دو کروموزوم زیر را داشته باشیم :

1 2 3 | 4 5 6

3 6 4 | 2 1 5

بدست خواهیم آورد :

v_1 = 1 2 3 | 6 4 5

v_2 = 3 6 4 | 1 2 5

crossover تا حالا تمرکز روی موقعیت شهر در دور را جستجو کرده در حالیکه در واقع یالها مهمترین قسمتهای سفر فروشنده دوره گرد هستند زیرا آنها وزنها را معین می کنند بنابراین چیزیکه ما واقعا می خواهیم این است که با یالها بیشتر از موقعیت هر شهر سروکار داشته باشیم.

Grefenstette (1981) یک روتین crossover اختراع کرده که هر راس را از یکی از آنهایی که برای راس جاری در یکی از parentها لازم است بر می دارد.

این را بوسیله ایجاد یک لیست یال برای هر راس انجام می دهیم. کروموزومهای:

v1 = 1 2 3 4 5 6

v2 = 3 6 4 2 1 5

ابتدا یکی از نخستین راسها را از یکی از parentها یعنی 1 یا 3 در این مثال انتخاب می کنیم آن را که کمترین عدد از راسهای لازم را دارد انتخاب می کنیم و یا اگر آنها عدد یکسانی دارند تصادفی یکی را انتخاب می کنیم.

سپس مشاهده می کنیم که راسهایی با راس 1 تلاقی می کنند. چون این همان راسی است که اول انتخاب کردیم دوباره راسی با کمترین عدد از راسهای لازم که قبلا انتخاب نشده اند انتخاب می کنیم سپس راس 2 را انتخاب می کنیم این فرایند انتخاب راسهای فرضی را ادامه می دهیم اگر به حالتی برخورد کنیم که نتوانیم راسی را که قبلا انتخاب نشده را انتخاب کنیم یک گره که قبلا انتخاب نشده است را تصادفی انتخاب می کنیم.

این به آن معنی است که ما می خواهیم راسی را بدست آوریم که در راس جاری ما در یکی از parent ها روی نداده است اما متاسفانه این اجتناب ناپذیر است بنا براین کروموزوهای parent ما می توانند فرزند تولید کنند.

توجه داشته باشید که ما در تمام مدت در توانایی انتخاب راس هایی که در یکی از parent ها لازم بودند موفق بودیم.

ما فقط یک فرزند از این crossover بدست می آوریم تا بسیاری از crossover ها را دوبار انجام دهیم تا یک نسل جدید ایجاد کنیم. همچنین عملگرهای crossover ای داریم که از اطلاعات مکشفه ای استفاده می کند.

Heuristic Crossover یک راس تصادفی برای شروع انتخاب می کند و سپس به دو یالی که راس جاری را در کروموزومهای parent رها می کند توجه می کند و کوتاهترین یالی را که یک دور را نشان نمی دهد بر می دارد. اگر هر دو یال یک دور را نشان دهند به طور تصادفی یالی را که این کار را انجام ندهد انتخاب می کنیم.

3. Mutation:

ابتدا عملگر دو گزینه ای (2 - opt) را نشان می دهیم. دو یال (a , b) و (c , d) را از دورمان انتخاب می کنیم و چک می کنیم که آیا می توانیم این 4 راس را با یک روش متفاوت به هم وصل کنیم تا کمترین وزن را به ما بدهد یا خیر. برای انجام این کار چک می کنیم که اگر $C_{ab} + C_{cd} > C_{ac} + C_{db}$ باشد یالهای (a , b) و (c , d) را با یالهای (a , c) و (d , b) عوض کنیم. توجه کنید که فرض کرده ایم که a و b و c و d با ترتیب مشخصی در دور ظاهر شده اند حتی اگر b و c متصل نباشند.

همچنین یک عملگر سه گزینه ای (3 - opt) داریم که به جای دو یال سه یال تصادفی را نشان می دهد. اگر یالهای (a , b) و (c , d) و (e , f) را داشته باشیم چک می کنیم که اگر $C_{ab} + C_{cd} + C_{ef} > C_{ac} + C_{be} + C_{df}$ باشد یالهای (a , b) و (c , d) و (e , f) را با یالهای (a , c) و (b , e) و (d , f) عوض کنیم.

عملگر (or - opt) شبیه (2 - opt) است.یک مجموعه از راس های متصل را تصادفی انتخاب می کنیم و چک می کنیم که آیا این رشته می تواند بین دو راس دیگر اضافه شود تا وزن تقلیل یابد یا خیر.

ما می توانیم این را بوسیله پیدا کردن مجموع وزن های یالهای اضافه شده و مجموع وزن های یالهای حذف شده محاسبه کنیم. اگر وزن یالهای حذف شده بیشتر بود تعویض انجام گیرد.

سه عملگر Mutation دیگر نیز وجود دارند که یک شهر انتخابی تصادفی را به یک مکان انتخاب شده تصادفی اضافه می کند.

همچنین ما هنگامیکه دو شهر تصادفی را انتخاب می کنیم و آنها را تعویض می کنیم یک mutaion دو جانبه داریم.

مقایسه روشهای مختلف الگوریتم ژنتیک برای TSP:

تا به حال فرمهای مختلفی از رمزگذاری ها encoding و عملگرهای crossover و mutation را در حل مساله TSP به روش الگوریتم ژنتیک دیدیم. این حالتها می توانند با هم ترکیب شوند و منجر به رسیدن به راه حل های مختلفی برای TSP به روش الگوریتم ژنتیک شوند. ولی از آنجایی که متدهای crossover روی encoding های خاصی عمل می کنند در نتیجه الگوریتمهای ژنتیک خیلی متفاوتی برای جستجو نداریم.

حال به بررسی الگوریتمهای ژنتیک محض یعنی بدون استفاده از Heuristic Information می پردازیم.

فرض کنید که PMX crossover را انتخاب کرده ایم و هیچ عملگری را برای mutation اتخاذ نکرده ایم. با این شرایط در 33 شهر به جوابی می رسیم که طول آن 10 درصد از جواب بهینه بیشتر است. و برای 100 شهر این میزان به 210 درصد می رسد. اگر در یک مساله که از 30 شهر تشکیل شده است اگر از PMX استفاده کنیم بهترین طول 498 و اگر از Order Crossover استفاده کنیم این میزان به 425 کاهش می یابد. در حالی که Cycle Crossover نتیجه ای برابر 517 می دهد.از آنجایی که می دانیم در این مساله خاص(30 شهر) بهترین جواب طولی برابر 420 دارد به نظر می رسد که Order Crossover جوابی بهتر از بقیه بر می گرداند.

حال به بررسی Matrix Crossover می پردازیم. اگر از یک crossover دو نقطه ای استفاده کنیم مشاهده می کنیم که برای 30 و 50 و 75 و 100 و 318 دورهایی با طول 420 و 426 و 535 و 629 و 42154 را ارائه می کند. که همه این جوابها کمتر از 2 درصد بیشتر از جواب بهینه هستند.

پس احتمالا استفاده از یالها بسیار امیدوار کننده تر از استفاده از راسها به عنوان متغیر است.

توجه کنید که به هر حال نمایش ماتریس فضای بیشتری را برای ذخیره کردن نسبت به نمایش به صورت عدد صحیح و crossover ساده می خواهد و در ضمن محاسبات crossover و mutation در ماتریس پیچیده تر و زمانبر تر است.

همچنین روش دیگری که تست شده اینست که ما از (2 - opt) برای mutation استفاده کنیم و از crossover استفاده نکنیم. این روش نیز جواب خوبی ارائه می دهد ولی جواب قبلی بهتر از این روش است. در ضمن برای وقتی که n را زیاد فرض می کنیم این روش جوابی مناسب ارائه نمی دهد.

Heuristic Algorithm نیز به جواب خوبی می رسد.

Heuristic Algorithm وقتی که با mutation(2 – opt) ترکیب می شود بهترین جواب را در مقایسه با متدهایی که تا به حال گفتیم برمی گرداند. به طوری که این جواب بسیار نزدیک به مقدار بهترین جواب است. البته این روش فضای زیادی را اشغال می کند و نیز وزن هر یال باید در جایی ذخیره شود.

در نتیجه می بینیم که الگوریتم ژنتیک وقتی که از نمایش ماتریس برای encoding و از Matrix Crossover یا Heuristic Crossover استفاده می کند بهترین جواب را برمی گرداند و بهتر از دیگر روشها کار می کند.

در هر دو روش crossover بالا استفاده از mutation(2 – opt) کیفیت الگوریتم را افزایش می دهد.

نتیجه گیری:

الگوریتمهای ژنیتیک به نظر می رسد که یک جواب خوب برای TSP پیدا می کند. در حالیکه کارایی الگوریتم به میزان زیادی به نحوه encode کردن و نیز crossover و mutation بستگی دارد. به نظر می رسد که استفاده از نمایش Matrix و

Heuristic Information بهتر از بقیه روشها کار می کند و جواب قابل قبولی را برمی گرداند که به جواب واقعی بسیار نزدیک است.

احتمالا الگوریتم ژنتیک روش بهتری نسبت به دیگر روشها برای TSP است ولی هنوز جواب بهتری نسبت به دیگر روشهای موجود پیدا نکرده است. ولی این را نیز می دانیم که بهترین الگوریتمهای غیر ژنتیکی ارائه شده برای TSP در حالتهای خاصی از الگوریتمهای ژنتیک ارائه شده است. پس ما امیدواریم که با ارائه روتین های بهتری برای Encoding و Crossover و Mutation راه حلهای مناسب تری برای Traveling Salesman Problem ارائه شود.

مراجع :

[1]. Kylie Bryant , Genetic Algorithm and the Traveling Salesman Problem , Hervey Mudd college , 2000

[2]. Dr Alex Rogers , CM2408 – Symbolic Al Lecture 8 – Introduction to Genetic Algorithms ,December 2002

[3]. Mehrdad Dianati , Insop Song , Mark Treiber , An Introduction to Genetic Algorithms and Evolution Strategies , University of Waterloo , Canada , 2002

[4]. Jean – Philippe , Ph.D , Genetic Algorithm Viewer : Demonstration of a Genetic Algorithm , May 2000

[5]. Eric Krevice Prebys , The Genetic Algorithm in Computer Science , 1997

[6]. Marcin L. Pilat and Tony White , Using Genetic Algorithms to Optimize ACS-TSP , School of Computer Science, Carleton University , Canada , 2000

[7]. Genetic Algorithm , Beasly – Bull – Martin , October 2000

[8]. Dr. George Magoulas , Some problems are too difficult to solve , CS3018S

[9]. Byung-In Kim , Jae-Ik Shim , Min Zhang , Comparison of TSP Algorithms , December 1998

[10]. Chia-Hsuan Yeh , Graduate Course An Introduction to Genetic Algorithms , Goldberg , 1989