بسم الله الرحمن الرحيم

دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

بررسی الگوریتم های تست نرم افزار

گزارش پروژه کارشناسی

مینا بیرامی -**۹۳۲۳۹۳۳**

استاد راهنما :

دکتر الهام محمودزاده

شهریور **۱۳۹۷**

فهرست

چکیده ۱

فصل اول : مقدمه ای بر تست اتوماتیک نرم افزار ۲

۱-۱مقدمه ۲

۱-۲ کارهای انجام شده ۳

فصل دوم: بررسی مفاهیم مرتبط ۷

۲-۱ گراف کنترل جریان(*Control Flow Graph*) ۷

۲-۲ الگوریتم های تکاملی : الگوریتم ژنتیک ۸

۲-۲-۱ ساز و کار الگوریتم ژنتیک: ۹

۲-۳ یادگیری ماشین(*Machine Learning*) ۱۱

۲-۳-۱ ماشین بردار پشتیبان (*Support Vector Machine*) ۱۳

۲-۳-۲ درخت تصمیم گیری(*Decision Tree*) ۱۴

فصل سوم : الگوریتم پیشنهادی ۱۶

۳-۱ توضیح مساله ۱۶

۳-۲ بخش اول تحقیق : الگوریتم ژنتیک ۲۴

۳-۲-۱ بخش اول : تولید جمعیت اولیه ۲۴

۳-۲-۲ بخش دوم:مطالعه ساختار برنامه ۲۵

۳-۲-۳ بخش سوم:ایجاد شرط خاطمه برای برنامه ۲۵

۳-۲-۴ بخش چهارم: تعریف تابع امتیاز دهی (*fitness function*) ۲۵

۳-۲-۵ بخش پنجم: تابع بازترکیبی(*crossover function*) ۲۶

۳-۲-۶ بخش ششم:تابع جهش (*mutation function*) ۲۶

۳-۲-۷ بخش هفتم :نتیجه ۲۶

۳-۳ بخش دوم تحقیق : الگوریتم های یادگیری ۲۷

۳-۳-۱ بخش اول :تعریف ساختار مسیر ۲۷
۳-۳-۲ بخش دوم:ماتریس سردرگمی (*Confusion Matrix*)..... ۲۷
۳-۳-۳ بخش سوم : دیتاست (*dataset*) ۲۹
۳-۳-۴ بخش چهارم : داده های تمرین و داده های تست (*Train/Test Data*) ۳۰
۳-۳-۵ بخش سوم :به کار گیری الگوریتم *svm*: ۳۰
۳-۳-۶ بخش چهارم : به کار گیری الگوریتم *decision Tree* :..... ۳۱
فصل چهارم :نتیجه گیری ۳۲
منابع ۳۳

چکیده

تست اتوماتیک نرم‌افزار برای شناسایی محصول میزان اعتماد به آن ،ما را از دردسر های تست سنتی دور می‌کند و از هزینه‌های جانبی جلوگیری می نماید.

در این پژوهش سعی بر آن داشتیم تا الگوریتم های مبتنی بر جست و جو و الگوریتم های یادگیری را برای تست اتوماتیک قطعه کد های کوچک اعمال کنیم. به همین منظور، برنامه ای را انتخاب کرده و با استفاده از گراف کنترل جریان آن ، مسیر های احتمالی را شناسایی کرده و با استفاده از الگوریتم ژنتیک به تولید کردن آن مسیر ها پرداختیم. هدف از این کار شناسایی مسیر های یک قطعه کد و تولید *test-case* هایی است منجر به پیمودن مسیر های برنامه می گردد.برای یافتن مسیر های صحیح یک بار از الگوریتم ژنتیک و یک بار دیگر از الگوریتم های یادگیری استفاده کردیم .سپس مسیر های تولید شده را جمع آوری کرده و الگوریتم های مبتنی بر جست و جو *SVM* و *Decision Tree* را برای طبقه بندی مسیر ها اعمال کردیم.

کلمات کلیدی: گراف کنترل جریان، تست اتوماتیک نرم افزار، الگوریتم های تکاملی ، الگوریتم ژنتیک در تست نرم افزار ، الگوریتم های یادگیری ماشین

فصل اول : مقدمه ای بر تست اتوماتیک نرم افزار

۱-۱مقدمه

تست اتو ماتیک نرم افزار و روش های سریع رسیدن به جواب در این تست ها ، همواره مورد توجه محققان در طول سال های متمادی بوده است.اما به دلیل پر هزینه بودن ، سخت و پیچیده بودن این عمل و نیاز میرم به یک آزمایشگاه مجهز، این راه را دشوار نموده است.با این اوصاف بیشتر از ۵۰ درصد توسعه ی یک پروژه نیر صرف تست کردن آن می‌شود . [1]با این اوصاف ، اتوماتیک کردن این روند می‌تواند تأثیر به سزایی در وقت و هزینه یک پروژه داشته باشد.متود های بیشماری در این راه استفاده شده است. خیلی ها به جواب نرسیده اند و بعضی از آنها مانند رندوم قابل اعتماد نیست.به عبارت دیگر، بسیاری از الگوریتم های پیشنهاد شده برای تست اتوماتیک به وسیله سایز و پیچیدگی محدود شده اند و باقی آنها ، یا به جواب لازم نمیرسند و با قابل اعتماد نیستند.[2]

تا به اینجای کار به این نتیجه رسیدیم که مساله تولید دیتا برای برنامه یک مساله غیر قابل تصمیماست.متاهیوریستیک های ارائه شده برای تست و تولید دینا ، جز مسائل *NP-Hard* طبقه بندی شده است.[2]

[2]قواعد تست کردن یک برنامه از گذشته تا به کنون تقریباً یکسان باقی‌مانده است . بدین صورت که معیار های مورد نظر تست[1] را به یک تابع هدف[2] تبدیل کنیم.الگوریتم های استفاده شده در حال حاضر از این قانون تبعیت می کند.اما خب این سؤال پیش می‌آید که این توابع هدف چگونه باید تعریف شوند؟برای جواب به این سؤال ابتدا باید سؤال اساسی‌تری پاسخ دهیم : تابع هدف چیست؟به طور کلی می‌توان پاسخ داد :تابعی که با مقایسه راه حل‌هایی که الگوریتم پیاده‌سازی شده به ما ارائه داده استُ مطابقت آن را با هدف‌های اصلی برنامه چک می کند است:

- تست مربوط به ساختار برنامه (*white-box testing*)
- تست مربوط به ویژگی های برنامه(*black-box testing)*

[1] *Test Citra*
[2] *Objective function*

- تست مربوط به امنیت برنامه(*gray-box-testing*)

در این پژو هش ، ما قصد داریم به بررسی برخی از الگوریتم های حل مساله مورد نظرمان ، به طور خیلی ویژه به الگوریتم ژنتیک (*GA*) بپردازیم ، مسیر های مختلفی که برنامه طی می کند را بررسی و روند تولید *test case* را برای آن به صورت اتوماتیک رقم بزنیم .با توجه به توضیحات مختصری که داده شد ، موضوع این پژوهش در حوزه *white-box* یا همان تست ساختاری است.ما ساختار برنامه را بررسی می کنیم و با قواعدی که از آن استناد میکنیم ، به مسیر هایی که به تولید داده های تست منجر می شوند را تولید کنیم.

۲-۱ کارهای انجام شده

مهم‌ترین بخش در تست یک برنامه انتخاب یک metaheuristic مناسب برای مساله است. پس ما باید با مساله خود به خوبی آشنا باشیم.پس از تعریف مساله ، الگوریتم هایی برای مساله باید درنظر گرفته شود .این الگوریتم با توجه به ساختاری که برنامه ما دنبال می‌کند ، تفسیر و ویژگی‌های مختص خود را داراست. ما توابع هدف را تعریف می‌کنیم و خروجی آن‌ها ، حکم یک مدرک [۳] دارد که در زبان رمز شده برنامه نویسی محبوس شده است. نتایج آن‌ها ، راه حل [۴] هایی هستند که با توجه به تابع هدف ، یا با هدف اصلی برنامه یکی هستند و یا در نزدیکی و مجاورت جواب مساله قراد دارد.[2]

روند کلی را می‌توان در ۳ گام زیر خلاصه کرد:

۱. *metaheuristic* مورد نظر انتخاب کنیم
۲. توابع هدف را تعریف کنیم
۳. از کاندید های *solution* ، میکنیم کدام در حوزه جواب ما هستند : یا کنار گذاشته می‌شوند (اگر در مجاورت هدف اصلی نباشد)و یا دستخوش تغییر می‌شوند (در صورتی که جواب باشند بدون تغییر باقی می مانند)

در این بخش میخواهیم به روند تکامل و پیدایش این الگوریتم ها بپردازیم و جایگاه آنها در تست اتوماتیک نرم افزار پیدا کنیم:

[3] *clue*
[4] *solution*

- **الگوریتم تپه نوردی(*Hill Climbing*)**

یکی از الگوریتم های مبتنی بر جست و جو محلی[5]، الگوریتم تپه نوردی است. محلی بودن به این معناست که اگر به عنوان مثال چندین راه حل اولیه در اختیار داشته باشیم ، با پرورش دادن و تغییر دادن یکی از آنها سعی در بهتر کردن آن و رسیدن به جواب را دارد. در حقیقت ، در محدوده ی(همسایگی) یکی از راه حل ها بازرسی می کند تا به جواب برسد(ممکن هم هست که نرسد).اگر راه حل بهتری در طی این بازرسی ها پیدا شد ، با راه حل فعلی جایگزین خواهد شد.به شبه کد زیر دقت کنید:

```
function HILL-CLIMBING( problem) returns a state that is a local maximum
   inputs: problem, a problem
   local variables: current, a node
                    neighbor, a node
   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      neighbor ← a highest-valued successor of current
      if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
      current ← neighbor
```

شکل ۱- ۱ شبه کد الگوریتم تپه نوردی *[18]*

جست و جوی محلی امکانی است که در بسیاری از برنامه ها می تواند ما را در رسیدن به جواب سرعت بخشد. چندین پروژه تست اتوماتیک سیستم برای تولید *test-case* از این الگوریتم بهره برده اند:

--سیستم مطرح شده ، تمامی تست کیس های مربوط به هر نوع کد *assembly* را پوشش می دهد. فرمت های مختلف یک تست کیس مانند *Nunit* و... را پشتیبانی می کند. متود به کار رفته برای تولید داده تست ، الگوریتم تپه نوردی است. تولید داده در تمامی سطوح تست (*black ,white ,gray*,..) بررسی و نتیجه آنها بررسی شده است.[3]

[5] *local*

الگوریتم تپه نوردی ساده است و سریع به جواب می رسد ولی قادر نیست محدوده های دیگر را برای جست و جو بررسی کند . به همین دلیل میزان درستی جوابی که به ما می دهد به شدت وابسته به راه حل ابتدایی(starting solution) است.

- **الگوریتم سردسازی شبیه سازی شده(*Simulated Annealing*)**

با توجه به میزان وابستگی الگوریتم تپه نوردی به راه حل ابتدایی ، محققان در پی الگوریتم مستقل تر از نقطه ی ابتدایی بودند. در طی سال ها الگوریتمی مشابه به تپه نوردی با این تفاوت کهبه جای بررسی یک نقطه به تمامی نقاط احتمال می دهد و به راه حل های ضعیف تر، احتمال کمتری اختصاص می دهد.[2]

$x \leftarrow x_0$
$e \leftarrow \mathrm{Energy}(x_0)$
$t \leftarrow \mathrm{Temperature}(0,\ \ldots)$
$k \leftarrow 0$
while stop criteria not met **do**
  $t \leftarrow \mathrm{Temperature}(k,\ \ldots)$
  Pick $x_{\mathrm{test}} \in \mathcal{N}_x$:
  $e_{\mathrm{test}} = \mathrm{Energy}(x_{\mathrm{test}})$
  if $P(e, e_{\mathrm{test}}, t) > \mathrm{Random}()$ **then**
    $x \leftarrow x_{\mathrm{test}}$
    $e \leftarrow e_{\mathrm{test}}$
  end if
  $k \leftarrow k + 1$
end while

شکل ۲- ۱ شبه کد الگوریتم سردسازی شبیه سازی شده*[19]*

طی پژوهش انجام شده ، این الگوریتم می تواند در یک مدت زمان معین تعداد خطا های بیشتری را در برنامه پیدا کند.بنابراین زمان کمتری جهت تست کردن نیاز است.[4]

نام این الگوریتم از فرآیند تغییر انرژی در سیستم بهره گرفته است . به این معنا که دمای آن کاهش می یابد تا به یک وضعیت پایدار تر برسد.[2]

- **الگوریتم های تکاملی(*Evolutionary Algorithms*) : الگوریتم ژنتیک**

الگوریتم های تکاملی ، از سری الگوریتم هایی هستند که با تغییر دادن و تکامل دادن جواب‌های کاندید برای مساله ، قصد رسیدن به هدف را دارد . الگوریتم ژنتیک ، از موفق ترین الگوریتم های تکاملی ، پس از گذشت سال ها همچنان در مقالات علمی و پژوهش ها نقش پررنگی را ایفا می کند. [2] الگوریتم ژنتیک یک الگوریتم

بهینه سازی[6] بر اساس پایه های بیولوژیکی طبقه بندی می شود.همچنین محدودیت قبلی از جمله جست و جوی محلی را ندارد و قابلیت جست و جوی جهانی[7] به معنای جست و جو در فضای گسترده تر را دارا می باشد. [5] به شبه کد زیر [6]دقت کنید:

```
(30) Test case set S = ∅
(31) for each coverage C do
(32)     Find start node, N_S
(33)     repeat
(34)         for (i = 0; i < |N_S| /2; i++) do
(35)             Select two parents in the population
(36)             Generate two offspring by crossover operation between two parents
(37)             Insert two offspring into new generation list
(38)             if a new offspring satisfy the coverage, C then
(39)                 S = S ∪ (Σ of the offspfing)
(40)                 break
(41)             end if
(42)         end for
(43)         Mutate some offspring in the new generation list
(44)     until satisfy C or reach maximum iteration
(45) end for
```

شکل ۳- ۱ شبه کد الگوریتم ژنتیک منبع :*[7]*

در پژوهش های انجام شده در خصوص تست اتوماتیک نرم افزار با استفاده از الگوریتم ژنتیک ، با ارائه ***fitness function*** های مختلف در جهت بهبود زمان تولید مسیر های تست برنامه تلاش کرده اند . به طور کلی به اثباتا رسانیده اند که تست اتوماتیک با استفاده از الگوریتم ژنتیک بر تست های تصادفی برای تولید داده تست را باید در الویت قرار داد تا نتایج بهتری حاصل شود. [5]

- **یادگیری ماشین(Machine Learning)**

با گذشت زمان و پیشرفت هر چه بیشتر داده کاوی[8] در تمامی زمینه ها ، تست اتوماتیک نیز از این تکنولوژی بهره گرفته و به این امر سرعت بخشیده است .از جمله پژوهش های مرتبط ، بررسی داده کاوی در متود های تست کردن white box و black box و مقایسه آنها می توان اشاره کرد. [8]

[6] *optimization*
[7] *global*
[8] *Data mining*

فصل دوم: بررسی مفاهیم مرتبط

در این فصل به طور ویژه به تحقیق در موردفاکتور هایی میپردازیم که در پیشبرد این پروژه ، تاثیر مستقیم داشته اند.به عبارتی یک سری مفاهیم اولیه تعریف میشود و در فصل بعدی با توجه به این مفاهیم ، کار پیش خواهد رفت.

۲-۱ گراف کنترل جریان(*Control Flow Graph*)

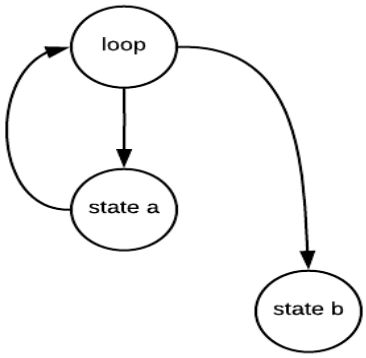
همانطور که می‌دانیم ، هر برنامه شامل تعدادی خط کد است . گراف کنترل جریان ، گرافی است که هر خط برنامه را یک گره [۹]در نظر میگیرد و هر گره را به گره‌هایی که در اجرای برنامه ملاقات می کند ،متصل می کند .در حقیقت به دو گره یک یال رسم می‌کند .این گراف برای آنالیز های آماری و ریاضی و بهینه سازی در کامپایلرها ، ضروری است.[9]

طبق پژوهشی که بر روی تست نرم افزار از گذشته تا به کنون انجام شده است، رسم گراف کنترل جریان برای قطعات کد در موارد زیر به ما کمک کند: [5]

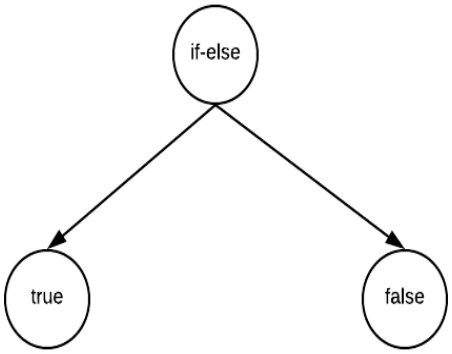
- تشخیص مسیر های مستقل در برنامه
- آماده دن داده ها برای اجرا شدن در تمامی مسیر های برنامه

به طور کلی در برنامه با دو نوع قطعه کد مواجه می‌شویم : جملات شرطی و حلقه ها. از نوع اتصال گره‌ها در گراف کنترل جریان متوجه می‌شویم که کدام جملات شرطی و کدام حلقه هستند . [9] به گراف های زیر توجه کنید:

[9] node



شکل ۱-۲ نمایی از یک حلقه-همانطور که مشاهده می شود، یک نود به یکی از دو وضعیت زیر می رود. اگر شرط خاتمه حلقه برقرار بود(state a) دوباره به حلقه(loop) باز می گردد در غیر این صورت به یکه وضعیت دیگر میرود(state b) *[9]*



شکل ۲- ۲ همانطور که مشاهده می شود، از یک نود دو فلش به سمت بیرون آمده است.به این معنا است که در یک مسیر خاص فقط یکی از نود های خارج شده از نود اصلی طی میشود(یا true و یا false) *[9]*

استخراج مسیر ، از مزایای رسم این گراف است . مسیر ، دنباله ای از گره‌ها است که از گره ابتدایی [10] شروع می‌شود به خط انتهایی برنامه[11] می‌رسد و در بین این دو گره ، تعدای گره میانی را شامل می شود. [2]البته لازم به ذکر است که در یک مسیر الزاماً نود های ابتدایی و انتهایی نباید ابتدا و انتهای یک گراف باشد اما بحث انجام شده در این پژوهش ، اجرای یک برنامه و مسیر هایی است که یکه برنامه احتمال دارد طی کند . پس حتماً از جایی خاص شروع می‌شود و در نقطه ی انتهایی پایان می یابد.

۲-۲ الگوریتم های تکاملی : الگوریتم ژنتیک

الگوریتم ژنتیک ، زیرمجموعه ی الگوریتم های تکاملی است . ریشه ی این الگوریتم از طبیعت ناشی می‌شود . موجودات برای تطبیق خود با شرایط موجود و به عبارتی زنده ماندن ، قصد بهتر کردن نوع خود را دارند . همانطور که در فصل اول گفته شد ، با تکامل جواب‌های کاندید قصد داریم به جواب یا جواب‌های مساله دست یابیم. [2]الگوریتم ژنتیک نیز از این قاعده مستثنا نیست . بنا براین باید کاندید های موردنظر را در ساختاری قرار دهیم و با متحول کردن اجزای آن ساختار در طی چندین مرحله به محدوده جواب‌های مساله برسیم.

ساختار الگوریتم ژنتیک عبارت است از :

- ژن : کوچکترین جز یک دنباله (رشته)

[10] Start node
[11] End node

- کروموزوم :دنباله ای(رشته ای) از ژن ها

کروموزوم ها لزوماً طول یکسانی ندارند . به عبارتی تمام کروموزوم هی یک مساله خاص الزاماً از تعداد ژن یکسانی برخوردار نیستند .مجموعه‌ای از کروموزوم ها، جواب‌های کاندید ما را تشکیل خواهد داد.حال این سؤال پیش خواهد آمد که ژن ها چه ماهیتی دارند؟ برای اینکه بتوانیم مسائلمان را روی همچین ساختاری مطابقت دهیم ، لازم است تا از اعداد استفاده کنیم. بسیاری از کروموزوم های مورد استفاده در مسائل مختلف از مدل دو دویی [12]طبعیت می کنند.به این معنا که یک رشته از اعداد ۰ و ۱ به دنبال هم ، مفهوم و مدلی از یک جواب کاندید را به ما نشان می دهد. با رمز شکنی ۰ و ۱ ها ، به جواب(جواب های) واقعی خواهیم رسید. با توجه به اینکه هدف این

1	2	3	5	6	
1	4	6			

شکل۳-۲ دو رشته با طول های متفاوت-ژن های با مقدار های متفاوت

1	0	1	1	1	0
0	0	0	1	1	1

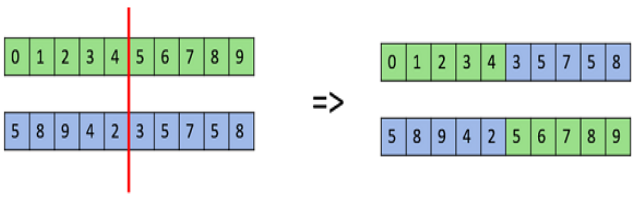
شکل ۴-۲ دو رشته هم اندازه با مقدار های 0 و 1

پژوهش تمرکز بیشتر بر روی الگوریتم ژنتیک است ، ما مسالمان را با هر دو ساختار دو دویی وعدد صحیح در نظر گرفتیم تا عمل‌کرد های هر کدام از این ساختار ها را جهت رسیدن به هدفمان امتحان کنیم .

۱-۲-۲ ساز و کار الگوریتم ژنتیک:

همانطور که گفته شد ، با تکامل کروموزوم های کاندید ، سعی داریم به جواب‌های موردنظر نزدیک شویم . ۳ عملیات ما را به این هدف نزدیک می کند : [5]

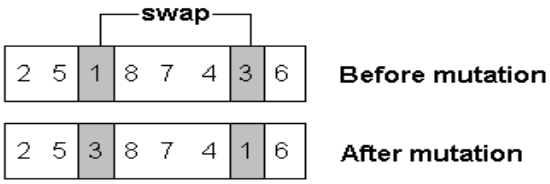
۱. بازترکیبی(*cross over*) : ژن های دو کروموزوم می‌توانند به چندین طریق مختلف با یکدیگر ترکیب شود و به اصطلاح فرزندی تولید کنند. برای این کار کافی است تا دو کاندید به صورت تصادفی انتخاب شوند و با هم ترکیب شوند . [5]یکی از نمونه های معروف باز ترکیبی در شکل زیر نمایان شده است:



شکل ۵-۲ نیمه اول رشته اول را با نیمه دوم رشته دوم جابجا میکنیم
منبع عکس: *[17]*

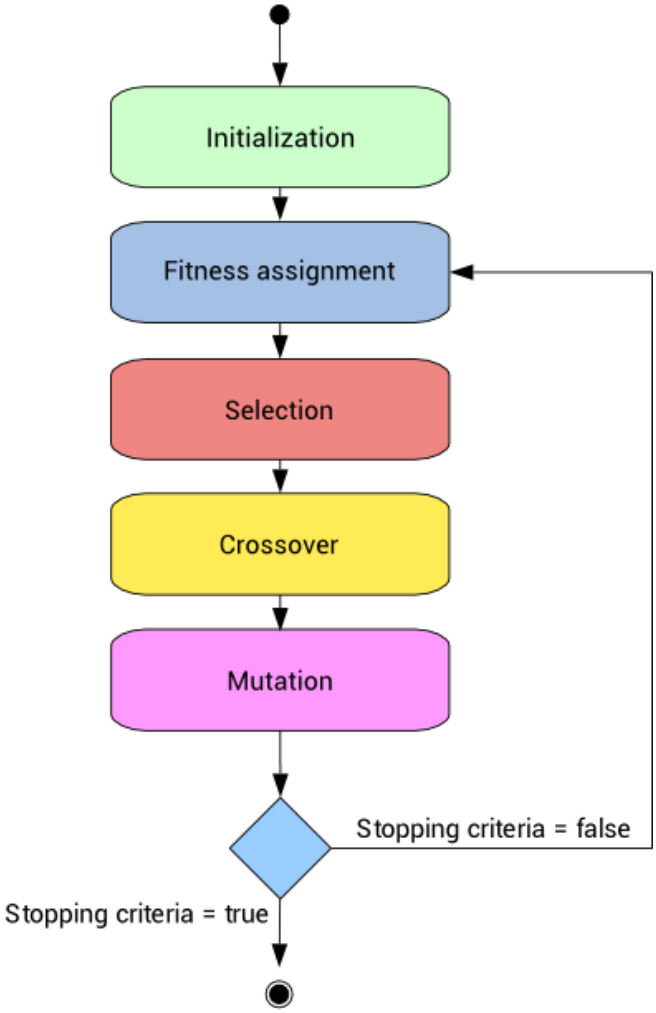
[12] binary

۲. جهش(*mutation*) : هر کدام از کروموزوم ها به تنهایی و بدون کمک کروموزم دیگری دچار تغییر ژنتیک می‌شوند .ژن های آن‌ها جابه جا و حذف خواهند شد. برای مثال به شکل زیر دقت کنید :



جهش تصادفی-دو ژن را به صورت رندوم با هم عوض میکنیم
منبع عکس: *[21]*

۳. انتخاب(*select*): بعد از عملیات جهش و بازترکیبی، از بین جمعیت جدید تولید شده ، بهترین ها انتخاب می‌شوند و برای عملیات در نسل بعدی آماده می شوند. معیار انتخاب بهترین کاندید ، تابع ارزیابی [۱۳]است که تصمیم می گیرد این کروموزوم جز کروموزوم های جمعیت جدید باشد یا خیر.



شکل ۶-۲ مراحل الگوریتم ژنتیک:
منبع شکل:*[20]*

اولین مرحله ، مقداردهی اولیه به جمعیت است .جمعیت تشکیل شده از افراد (*individual*) هاست که معمولا به صورت تصادفی مقدار دهی می شوند. دومین مرحله ، انتساب یک معیار برای سنجش افراد جمعیت است . کدام یک برتری دارند و کدام یک شانس کمتری نسبت به بقیه دارند.

در مراحل بعد با تعریف عملگر های انتخاب ، بازترکیبی و جهش که به آن پرداخته شد ، سعی داریم تا به محدود جواب مورد نظر دست پیدا کنیم.

[13] *Evaluation function*

۳-۲ یادگیری ماشین(***Machine Learning***)

امروزه ، تصور اینکه که یه ماشین بتواند ساز و کار چیزی را در یابد ، بدون این که برنامه نویسی شده باشد ، تحقیقات گسترده ای از پژوهشگران را دربر می گیرد. یادگیری ماشین (ml) در تمامی زمینه هایی که داده ای برای پردازش موجود باشد کاربرد دارد . همانطور که میدانیم ، امروزه دیگر مشکل ، کمبود داده نیست ،دقیقا برعکس : افزایش داده هایی که مرتب تولید می شودند(big data) ، پردازش آنها را بسیار سخت کرده است . بنابراین برنامه نویسی کردن برای تمامی این داده ها ، ممکن است بسیار زمانگیر و هزینه بر باشد . ممکن است این فکر به ذهن آدم برسد که می توان این داده ها را به قسمت های کوچک تر تقسیم کرد و برای قسمت های کوچک برنامه نویسی کرد . با این حال ممکن است نتیجه ای که از قسمت کوچکی از داده ها بدست می آید ، چندان قابل اعتماد نباشد. [10]

در گذشته نزدیک ، محققان به دنبال ساخت هوش مصنوعی برای یادگیری از داده های تولید شده بودند. داده هایی که بتوانند از آنها یک مدل تولید کنند. این مدل تولید شده که معمولا با یک رابطه ریاضی توصیف می شود ، توصیف گر مقدار زیادی داده است. ماشین[4] های بدون راننده ، سیستم های پیشنهاد گر در سایت های خرید ، تشخیص چهره افراد و ... همگی نتیجه تحقیقات در چند دهه اخیر بوده است. [10]

دو روش اصلی در یادگیری ماشین به کارگرفته می شود: [11]

- **یادگیری نظارت شده (*supervised learning*)**

الگوریتم هایی در این حوزه قرار می گیرند که به اصطلاح داری برچسب[14] باشند:

برای مثال فرض کنید با توجه به داده هایی که در اختیار داریم ، یک مدل از آن می سازیم. مدل ها یا در دسته ۰ قرار می گیرند یا در دسته ۱ . دو کلاس مختلف در این مدل تشکیلی شده است .هر کدام از این کلاس ها با یک برچسب (۰ و ۱)شناخته می شوند .نکته مهمی که در این روش به آن دقت کرد این است که حتما تعداد و نوع برچسب ها مشخص شده باشد .برای مثال اگر در تعدادی عکس ، حیوان های سگ ، گربه مشهود باشد ، با توجه به هر عکس ،رچسب های سگ و یا گربه به داده ها انتساب داده می شود .

الگوریتم هایی که در این روش مورد اتفاده قرار می گیرند عبارت است از :[12]

[14] *label*

- *Support Vector Machine(SVM)*
- *Decision Tree*
- *Gaussian Processes*
- *...*

- **یادگیری بدون نظارت(unsupervised learning)**

بر خلاف روش نظارت شده ، این روش تعداد و نوع برچسب ها از قبل مشخص نیست .الگوریتم انتخاب شده باید خودش تعداد کلاس ها و برچسب ها را تشخیص دهد و مدلی از روی آن بسازد تا بتواند داده های جدید را در آن کلاس ها قرار دهد یا اگر در هیچ کدام از کلاس ها قرار نگرفتند ، آنها را در کلاسی جدید بگذارد یک برچسب به آن اختصاص بدهد. [11]

برای مثال فرض کنید یک تعداد زیادی عکس از حیوانات مختلف داریم . با توجه به ویژگی های ظاهری هر کدام الگوریتم کلاسی به آن منتسب می کند و حیوانات با ویژگی های مشابه را در آن کلاس می گذارد.

از معروف ترین الگوریتم های استفاده شده در روش بدون نظارت عبارت است از:[12]

- *K-means*
- *mean-shift*
- *DBSCAN*
- *....*

روشی میان روش نظارت شده و بدون نظارت وجود دارد که به آن نیمه نظارت شده یا*Semi-Supervised* می گویند.یعنی بعضی از داده ها با یک برچسب مشخص شده اند و برخی دیگر خیر. به همین دلیل است که الگوریتم ها و متود های هر دوروش ذکر شده در بالا می تواند برای این روش استفاده شود. [11]

داده ها، در یک یا چند فابل به نام دیتاست [15]جمع آوری می شوند.هر داده ای به یک سری از ویژگی هایش شناخته میشود .هر ستون در دیتاست ، نمایانگر یک ویژگی از آن است .اگر از یک دیتاست آماده استفاده شود ، ویژگی ها از قبل مشخص شده اند و تنها کار آنالیز و به کارگیری الگوریتم های مختلف بر روی آن باقی می ماند . اما اگر

[15] *dataset*

قصد تهیه یک دیتا ست داشته باشیم با شناسایی مساله ویژگی های آن را مشخص میکنیم و آنالیز را بر روی آن آغاز میکنیم . با جزییات بیشتر در فصل بعدی به این موضوع خواهیم پرداخت.

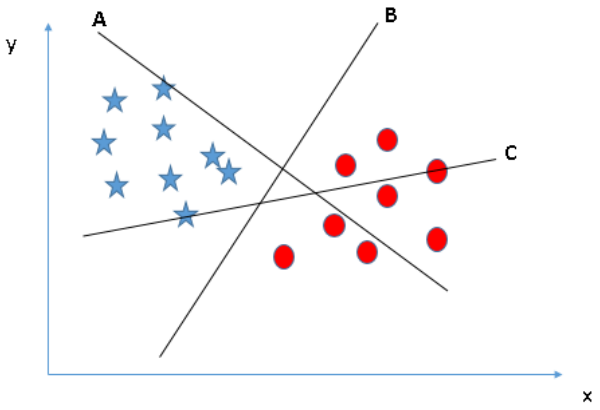
مساله ای در این پژوهش بر روی آن مانور داده ایم در فصل بعدی به دقت شرح داده میشود و دو مورد از مهم ترین الگوریتم های طبقه بندی[16]بر روی آن اعمال میشود و نتایج بررسی می گردد. برای آشنایی بیشتر ، توضیح مختصری از این الگوریتم ها و ساز کار آنها در زیر آمده است:

۱-۳-۲ ماشین بردار پشتیبان (*Support Vector Machine*)

همانطور که بخش بالا ذکر شد ، یکی از روش های یادگیری نظارت شده ، *SVM*[17] است . این روش برای طبقه بندی(*classification*) و تشخیص اوتلیر ها [18] به کار گرفته می شود.

۲-۳-۱-۱ ساز و کار **SVM**:

فرض کنید در طبقه بندی شما دو کلاس دایره و ستاره وجود دارد ، ما میخواهیم خطی را پیدا کنیم کهاین دو کلاس را از هم جدا کند.برای این کار لازم است که چندین خط رسم کنیم و بهترین آن را برگزینیم .مثلا در شکل زیر ، خط B ، بهتر از دو خط دیگر دو کلاس را تقسیم بندی کرده است.[13]



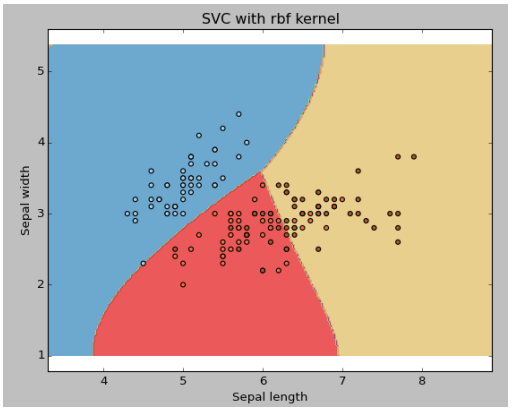
شکل ۲-۷ در یک صفحه بی شمار خط وجود دارد: الگوریتم چند خط به صورت رندوم رسم می کند و بهترین آن به عنوان جداکننده دو کلاس مشخص می شود.*[13]*

مثالی که در بالا مطرح شد یکی از ساده ترین مثال هایی است که *SVM* در آن کاربرد دارد. چرا که ممکن است داده های ما در صفحه یا فضا های ۳ بعدی و بیشتر(تعداد بعد فضای مطرح شده همان تعداد ویژگی ها یا *feature*

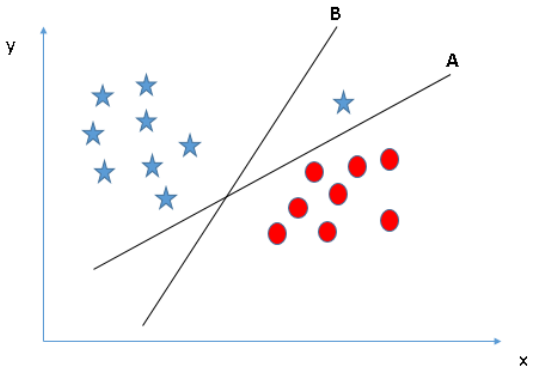
[16] *classification*
[17] *Support Vector Machine*
[18] *outliers detection*

های یک دیتا ست است) با یک خط ساده از هم جداپذیر نباشند . یا این تعداد کلاس ها بیشتر از ۲ باشد . :[13]

برای درک بهتر مسائلی به اشکال زیر دقت کنید



شکل ۸-۲ تعداد کلاس ها اگر بیشتر از ۲ باشد – در این صورت یک خط صاف نمیتواند آنها را از هم جدا کند *[13]*



شکل ۹-۲ به نظر می رسد خط A هر دو کلاس را از هم جدا کرده اما شاید کلاس ستاره دارای داده پرت باشد مانند ستاره ی سمت راست.بنا براین خط B جداکننده بهتری است *[13]*

۲-۳-۲ درخت تصمیم گیری(***Decision Tree***)

یکی دیگر از الگوریتم هایی که در حوزه روش های نظارت شده مطرح شد ، درخت تصمیم گیری است. بیشترین کاربرد آن در مسائل طبقه بندی است. به همین دلیل برای مساله ما یک الگوریتم کاملا مناسب است.مهم ترین کاربرد این الگوریتم ساخت یک مدل آموزش یافته برای دست یابی به قانون های یک دیتاست است.

مزیت عمده ای که باعث می شود از این الگوریتم استفاده شود ، مرحله به مرحله بودن آن است : به همانند انسان که از مرحله به مرحله فکر خود را گسترش می دهد تا به نتیجه برسد . بنابراین به دلیل وجود منطق ذهن انسان ، ارتباط گرفتن ما با آن بسیار راحت تر است. [14]

به عنوان مثال از طریق اطلاعات هواشناسی یک فایل با ویژگی های شکل زیر بدست آمده است . میخواهیم بدانیم با توجه ویژگی های مختلف آب و هو ، آیا بازی کردن گلف مناسب است یا خیر . با استخراج ویژگی هوایی و الویت بندی ویژگی ها می توان قوانین را به دست آورد . [15]و سوال مهم پیش خواهد آمد که : از کجا متوجه شویم که مهمترین ویژگی کدام است . ویژگی مهم بعدی کدام است و.... اگر تعداد ویژگی های یک دیتا ست بیشتر باشد ، این امر سخت تر و سخت خواهد شد .

معیار مهمی که ما را در این امر یاری میدهد information gain است : هر کدام از ویژگی هایی که اطلاعات بیشتری در اختیار ما قرار دهد ، عدد بالاتری را در IG خود باز می گرداند. به معادله زیر دقت کنید.

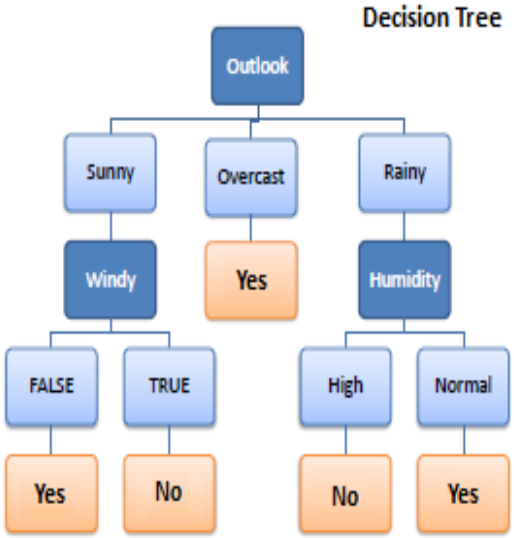
$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

معادله ۱ میزان بی نظمی یا همان Entropy در ویژگی های هر چقدر کمتر باشد، مارا کمتر دچار مشکل می کند. بنابراین میزان بی نظمی کل مجموعه اگر از یک میزان بینظمی یکی از ویژگی ها که مقدار کمی دارد کم شود ، طبق این فرمول gain بیشتری به ما می دهد.

این محاسبات برای هر کدام از ویژگی ها جداگانه انجام می شود و بالاترین Gain به عنوان ریشه درخت در نظر گرفته میشود . به این ترتیب ویژگی های بعدی سطح های بعدی درخت را می سازند تا به برگ که همان نتیجه است ، درسترسی پیدا کنند. به شکل زیر دقت کنید :

Predictors | Target

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No



شکل ۱۱-۲ مهم ترین ویژگی در داده های جدول بالا ، outlook است . برای همین باید به عنوان ریشه درخت درنظر گرفته شود. منبع شکل *[15]*

فصل سوم : الگوریتم پیشنهادی

در این بخش ، قدم به قدم به تولید مسیر های طی شده درمساله می پردازیم . این فصل شامل ۳ بخش اساسی است :

- توضیح و بررسی مساله و طراحی ساختار برای نگهداری گره های برنامه با استفاده از گراف کنترل جریان
- طراحی ساختاری برای مسیر ها و تولید آنها به وسیله تغییر دادن مسیر های رندوم و تولید مسیر های هدف با طراحی عملگر های الگوریتم ژنتیک
- طراحی دیتا ست نگهداری مسیر های تصادفی و مسیر های هدف و طبقه بندی آنها با الگوریتم های درخت تصمیم گیری و ماشین بردار پشتیبان

۳-۱ توضیح مساله

مساله می‌تواند هر برنامه‌ای با هر تعداد خط باشد . اما باید توجه داشت که الگوریتم ژنتیک به علت طبیعتش ، الگوریتم کم سرعتی است . به همین دلیل هر چقدر تعداد خطوط آن کمتر باشد ، تعداد مسیر هایی که یک برنامه می‌تواند تولید کند کاهش می‌یابد : به تبع آن ، الگوریتم در زمان کوتاهتری به جواب می‌رسد.

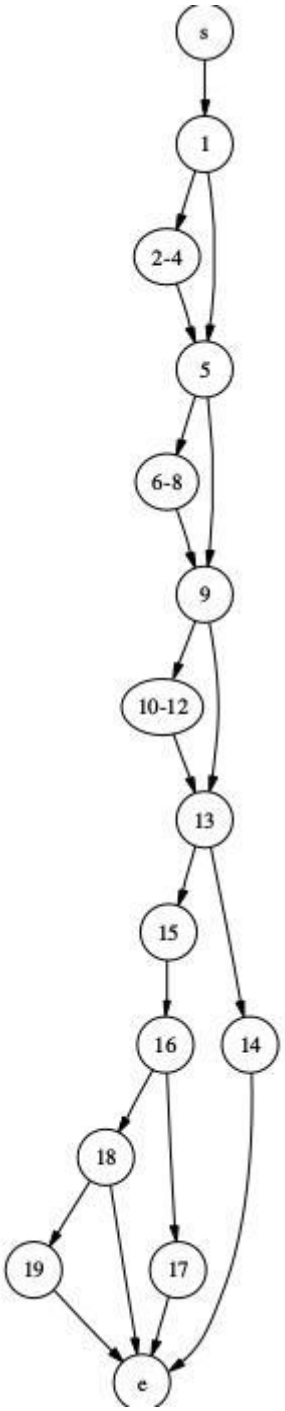
این پژوهش بر روی یک کد **تشخیص نوع مثلث** انجام گرفته شده است. از ویژگی‌های آن ، سادگی، خطوط کافی و عدم وجود هر گونه حلقه (while, for,...) در آن است.انتخاب این برنامه صرفاً به دلیل به تحقق رساندن هدف این پژوهش ،یعنی تست اتوماتیک نرم‌افزار است. به این قطعه کد توجه کنید:

```
 s     int tri_type(int a, int b, int c)
       {
           int type;

 1         if (a > b)
2-4        {   int t = a; a = b; b = t;   }

 5         if (a > c)
6-8        {   int t = a; a = c; c = t;   }

 9         if (b > c)
10-12      {   int t = b; b = c; c = t;   }

 13        if (a + b <= c)
           {
 14            type = NOT_A_TRIANGLE;
           }
           else
           {
 15            type = SCALENE;
 16            if (a == b && b == c)
               {
 17                type = EQUILATERAL;
               }
 18            else if (a == b || b == c)
               {
 19                type = ISOSCELES;
               }
           }

 e         return type;
       }
```

شکل ۱-۳ قطعه کد مربوط به تشخیص نوع مثلث: منبع شکل: *[2]*

این برنامه مشخص میکند که ۳ عدد داده شده ایا قابلیت تشکیل مثلث دارند یا خیر و اگر دارند ، نوع آن چیست.

با توجه به مطالب گفته شده درفصل قبل قسمت ۱-۲ در مورد گراف های کنترل جریان ، برای این قطعه کد ، گراف مورد نظر آن را رسم می‌کنیم تا تمامی مسیر هایی که احتمال دارد تا پایان اجرای برنامه در آن طی شود ، کمی برایمان واضح تر گردد.بنابراین داریم:



شکل ۲-۳ گراف کنترل جریان
(cfg) کد تشخیص نوع مثلث
منبع شکل : *[2]*

- ۲ انشعاب در نود شماره ۱
- ۲ انشعاب در نود شماره ۵
- ۲ انشعاب در نود شماره۱۳
- ۲ انشعاب در نود شماره۱۶
- ۲ انشعاب در نود شماره ۱۸

$$2 * 2 * 2 * 2 * 2 = 32$$

با توجه به انشعاباتی که در مسیر گراف دیده می‌شود، می‌توانیم ۳۲ مسیر از آن استخراج کنیم:

1 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 20]
2 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 20]
3 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20]
4 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 20]
5 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 14, 20]
6 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 15, 16, 17, 20]
7 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 15, 16, 18, 19, 20]
8 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 15, 16, 18, 20]
9 : [0, 1, 2, 3, 4, 5, 9, 10, 11, 12, 13, 14, 20]
10 : [0, 1, 2, 3, 4, 5, 9, 10, 11, 12, 13, 15, 16, 17, 20]
11 : [0, 1, 2, 3, 4, 5, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20]
12 : [0, 1, 2, 3, 4, 5, 9, 10, 11, 12, 13, 15, 16, 18, 20]
13 : [0, 1, 2, 3, 4, 5, 9, 13, 14, 20]
14 : [0, 1, 2, 3, 4, 5, 9, 13, 15, 16, 17, 20]
15 : [0, 1, 2, 3, 4, 5, 9, 13, 15, 16, 18, 19, 20]
16 : [0, 1, 2, 3, 4, 5, 9, 13, 15, 16, 18, 20]
17 : [0, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 20]
18 : [0, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 20]
19 : [0, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20]
20 : [0, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 20]
21 : [0, 1, 5, 6, 7, 8, 9, 13, 14, 20]
22 : [0, 1, 5, 6, 7, 8, 9, 13, 15, 16, 17, 20]
23 : [0, 1, 5, 6, 7, 8, 9, 13, 15, 16, 18, 19, 20]
24 : [0, 1, 5, 6, 7, 8, 9, 13, 15, 16, 18, 20]
25 : [0, 1, 5, 9, 10, 11, 12, 13, 14, 20]
26 : [0, 1, 5, 9, 10, 11, 12, 13, 15, 16, 17, 20]
27 : [0, 1, 5, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20]
28 : [0, 1, 5, 9, 10, 11, 12, 13, 15, 16, 18, 20]
29 : [0, 1, 5, 9, 13, 14, 20]
30 : [0, 1, 5, 9, 13, 15, 16, 17, 20]
31 : [0, 1, 5, 9, 13, 15, 16, 18, 19, 20]
32 : [0, 1, 5, 9, 13, 15, 16, 18, 20]

شکل ۳-۱۳ مسیر های استخراج شده از گراف کنترل جریان

اما تمامی این ۳۲ مسیر اتفاق نخواهد افتاد . چرا که شروطی که در برنامه ذکر شده است ، در مواقعی خاص هیچ گاه رخ نخواهند داد.به جدول زیر دقت کنید :

1	T T T	0,1,2,3,4,5,6,7,8,9,10,11,12,13,...
2	T T F	0,1,2,3,4,5,6,7,8,9,13,...
3	T F T	0,1,2,3,4,5,9,10,11,12,13,…
4	T F F	0,1,2,3,4,5,9,13,...
5	F T T	0,1,5,6,7,8,9,10,11,12,13,...
6	F T F	0, 1, 5,6,7,8,9,13,
7	F F T	0, 1, 5,9,10,11,12,13,
8	F F F	0, 1, 5,9,13,

جدول ۱ پشت سر هم گذاشتن شروط خطوط ۱ ، ۵ ، ۹ و حالت مختلف آنها -تمام مسیرهایی که میتواند به صورت منطقی اتفاق بیفتد در این جدول آمده است. نود های پیموده شده را در هر کدام از این شرایط می توان مشاهده می شود.

از نود شماره ۱ الی ۱۳ ، سه شرط اساسی بررسی می‌شود که در حقیقت ورودی های برنامه را کنترل کرده و بزرگترین آن‌ها رو در متغیر c ، کوچکترین آن‌ها را در متغیر a و دیگری که بین aو c است را در متغیر b قرار می دهد.هر کدام از این ۳ شرط می‌تواند درست باشد یا نقض شود .به عبارتی برای هر شرط ۲ حالت وجود دارد

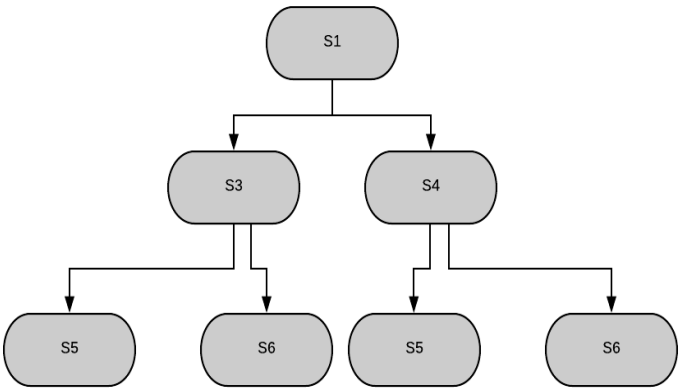
بر طبق جدولی ترتیب داده ایم تا بتوانیم روابط ریاضی هر کدام از ورودی های a و b و c را استخراج کنیم.

جدول ۲ روابط ریاضی داده های ورودی

شماره	وضعیت شرط	شماره خط	رابطه ریاضی
S1	شرط درست بوده	1	a>b ^ a'=b ^ b'=a
S2	شرط نقض شده	1	a<=b ^ a'=a ^ b'=b
S3	شرط درست بوده	5	a>c ^ a''=c ^ c'=a'
S4	شرط نقض شده	5	a'<=c ^ a''=a' ^ c'=c
S5	شرط درست بوده	9	b'>c' ^ b''=c' ^ c''=b'
S6	شرط نقض شده	9	b'<=c' ^ b''=b' ^ c''=c'

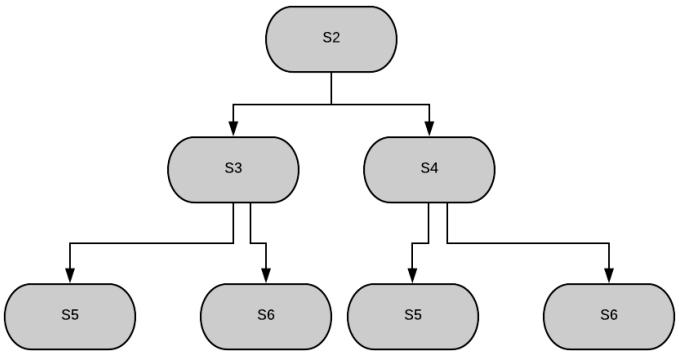
به دیاگرام های زیر دقت کنید : بنا به درست بودن یا نقض شدن هر کدام از شرایط در خطوط ۱ و ۵ و ۹ ، یک وضعیت[19] در نظر گرفتیم . ۸ حالت استخراج شده در شکل های ۴-۳ و ۵-۳ قابل مشاهده است.

2*2*2=8



شکل ۴-۳ چهار حالت ایجاد شده در صورت برقراری S1

S1-S3-S5
S1-S3-S6
S1-S4-S5
S1-S4-S6



شکل ۵-۳ چهار حالت ایجاد شده در صورت برقراری S2

S2-S3-S5
S2-S3-S6
S2-S4-S5
S2-S4-S6

با توجه به جدول ۱ و روابط ریاضی ، امکان رخداد هرکدام از آن ها را می خواهیم بررسی کنیم:(جدول در صفحه بعد نیز ادامه پیدا می کند)

[19] state

جدول۳ جدول بررسی تناقضات روابط ریاضی داده های ورودی

#	وضعیت	مسیر	روابط ریاضی	تناقض
1	T T T	0,1,2,3,4,5,6,7,8,9,10,11,12,13,...	a>b ^ a'=b ^ b'=a a'>c ^ a''=c ^ c'=a' b'>c' ^ b''=c' ^ c''=b'	✓
2	T T F	0,1,2,3,4,5,6,7,8,9,13,...	a>b ^ a'=b ^ b'=a a'>c ^ a''=c ^ c'=a' b'<=c' ^ b''=b' ^ c''=c'	× a=3 b=2 c=1
3	T F T	0,1,2,3,4,5,9,10,11,12,13,…	a>b ^ a'=b ^ b'=a a'<=c ^ a''=a' ^ c'=c b'>c' ^ b''=c' ^ c''=b'	✓
4	T F F	0, 1, 2,3,4,5,9,13,	a>b ^ a'=b ^ b'=a a'<=c ^ a''=a' ^ c'=c b'<=c' ^ b''=b' ^ c''=c'	× a=2 b=1 c=3
5	F T T	0,1,5,6,7,8,9,10,11,12,13,...	a<=b ^ a'=a ^ b'=b a>c ^ a''=c ^ c'=a' b'>c' ^ b''=c' ^ c''=b'	✓
6	F T F	0, 1, 5,6,7,8,9,13,	a<=b ^ a'=a ^ b'=b a>c ^ a''=c ^ c'=a' b'<=c' ^ b''=b' ^ c''=c'	× a=2 b=3 c=1
7	F F T	0, 1, 5,9,10,11,12,13,	a<=b ^ a'=a ^ b'=b a'<=c ^ a''=a' ^ c'=c b'>c' ^ b''=c' ^ c''=b'	✓

8	F F F	0, 1, 5,9,13,	a<=b ^ a'=a ^ b'=b a'<=c ^ a''=a' ^ c'=c b'<=c' ^ b''=b' ^ c''=c'	✓

طبق استدلال های ریاضی جدول ۳، TFF, TTF و FTF اتفاق نخواهند افتاد .بنابراین مسیر های باقی‌مانده عبارت است از:

1 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 20]
2 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 20]
3 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20]
4 : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 20]
5 : [0, 1, 2, 3, 4, 5, 9, 13, 14, 20]
6 : [0, 1, 2, 3, 4, 5, 9, 13, 15, 16, 17, 20]
7 : [0, 1, 2, 3, 4, 5, 9, 13, 15, 16, 18, 19, 20]
8 : [0, 1, 2, 3, 4, 5, 9, 13, 15, 16, 18, 20]
9 : [0, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 20]
10 : [0, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 17, 20]
11 : [0, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20]
12 : [0, 1, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15, 16, 18, 20]
13 : [0, 1, 5, 9, 10, 11, 12, 13, 14, 20]
14 : [0, 1, 5, 9, 10, 11, 12, 13, 15, 16, 17, 20]
15 : [0, 1, 5, 9, 10, 11, 12, 13, 15, 16, 18, 19, 20]
16 : [0, 1, 5, 9, 10, 11, 12, 13, 15, 16, 18, 20]
17 : [0, 1, 5, 9, 13, 14, 20]
18 : [0, 1, 5, 9, 13, 15, 16, 17, 20]
19 : [0, 1, 5, 9, 13, 15, 16, 18, 19, 20]
20 : [0, 1, 5, 9, 13, 15, 16, 18, 20]

شکل ۶-۳ مسیر های باقی مانده بعد از حذف تناقضات موجود در جدول ۳

طراحی دو ساختمان داده ی گراف گونه برای برنامه ، هدف اساسی بخش۱-۳ محسوب می شود :

- گراف اتصالات : یک گراف تشکیل شده از گره ها از ۰ تا ۲۰ که اتصالات هر گره را در خود نگه می دارد.
- گراف کنترل جریان :گراف کنترل جریان ، مسیر ها را در خود نگه می دارد .

۲-۳ بخش اول تحقیق : الگوریتم ژنتیک

همانطور که مطرح شد ، در بخش اول از تحقیق ، قصد داریم با استفاده از الگوریتم ژنتیک به تولید مسیر های درست بپردازیم.

۱-۲-۳ بخش اول : تولید جمعیت اولیه

در ابتدا ساختاری برای مسیرمان تعریف می کنیم : هر رشته(مسیر) شامل نودهایی است که ملاقات می کند. به عنوان مثال به شکل زیر دقت کنید:

0	1	5	9	13	15	16	18	20

0	1	5	9	13	14	20

شکل ۷-۳ رشته هایی با طول متفاوت- ابتدای آن ها ۰ و انتهای آن ها ۲۰ می باشد .

با توجه به شکل ۷-۳ ، دو مسیر با طول های متفاوت و هر گره ای که در برنامه ملاقات شده باشد ، به عنوان یک ژن برای ساختار در نظر گرفته می شود. بنابراین جمعیتی با تعداد محدود با طول های محدود به صورت تصادفی تولید میکنیم. به عنوان مثال ما معیت اولیه را ۵۰ در نظر گرفتیم . در تولید اعداد رندوم برای مسیر اولی به چند نکته باید توجه کرد:

- در این گراف کنترل جریان هیچ حلقه ای مشاهده نمی شود .چرا که از هیچ گره ای به گره یا گره های قبل خود رجوع نکرده است. بنا براین اعداد رندوم تولید شده باید روند صعودی داشته باشد: به عنوان مثال اگر عدد رندوم تولید شده برابر با ۵ باشد ، عدد رندوم تولید شده باید در بازه(5,20) باشد.
- این برنامه یک شروع و یک پایان دارد : یعنی الزاما از گره ۰ شروع می شود و الزاما در گره ۲۰ به پایان می رسد . بنابراین کوتاهترین مسیر رندوم تولید شده باید حداقل طول ۲ را دارا باشد.(0,20)

۳-۲-۲ بخش دوم:مطالعه ساختار برنامه

با توجه به گراف کنترل جریان برنامه در بخش قبل ، می توانیم آنالیز صحیحی از نقاط حساس برنامه داشته باشیم :

- بار اصلی برنامه بر نود شماره ۱۳ می باشد . تمامی مسیر ها بدون استثنا از این گره عبور می کنند
- گره های ۱ و ۵ و ۹ حتما در مسیر حضور دارند.
- گره های ۱ و ۵ و ۹ جملات شرطی هستند: به این معنا که در صورت برقراری آنها (نقض نشدن شرط)چند گره ی بعدی که در داخل scope شرط قرار می گیرند ، باید طی شوند . به عنوان مثال اگر شرط گره ۱ صحیح باشد ، نود های ۲ و ۳ و ۴ حتما طی خواهند شد.
- اگر گره های ۱۴ ، ۱۷ و یا ۱۹ در مسیر باشد ، هیچ گره دیگری به جز گره آخر(گره شما ۲۰) در مسیر نخواهد بود
- ...

بنابراین ما جمعیت های اولیه به صورت تصادفی را تولید می کنیم و با مطالعه ساختار برنامه ای که قصد تست کردن آن را داریم ، آن را هوشمند می سازیم. می توان اذعان داشت یک جهش هوشمند در ژن های هر رشته ایجاد می کنیم تا کاندیدهای مساله به جواب های اصلی نزدیک گردند. به صورت دقیق تر در بخش ۶-۲-۳ به آن خواهیم پرداخت.

۳-۲-۳ بخش سوم:ایجاد شرط خاطمه برای برنامه

در اینجا شرط خاتمه ما ، رسیدن به ۲۰ مسیر صحیح ذکر شده در بخش قبل می باشد .بنابراین ما مسیر های صحیح تولید شده توسط الگوریتم را جمع آوری می کنیم و زمانی که تمامی مسیر ها تولید شدند ، برنامه به اتمام میرسد

٤-۲-۳ بخش چهارم: تعریف تابع امتیاز دهی (*fitness function*)

برای این که بتوانیم از میان کاندید های مساله ، کاندید های برتر را برگزینیم ، نیاز به یک تابع امتیازدهی داریم: تابعی که به هر رشته عددی را انتساب دهد و هر چقدر این عدد بزرگتر باشد به این معنا است که این رشته ، رشته ی مناسب تری خواهد بود.

در ابتدا *fitness function* به صورت تعریف شده بود که صرفا یک اتصال درست در گراف اصلی ، یک امتیاز محسوب شود .ولس ئس از انجام دادن تست روی برنامه به نتیجه مطلوب نرسیدیم.

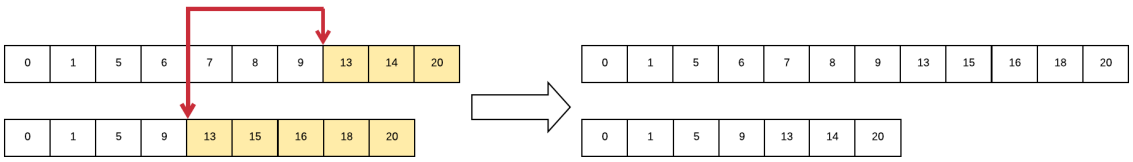
سپس ،تابع امتیاز دهی ما به این گونه تغییر یافت :

اگر طول رشته ما (مسیر) برابر با n باشد ، به ازای هر نودی که به درستی به گره بعدیش اتصال داشته باشد ، به اندازه $1/n$ به مقدار fitness آن افزوده خواهد شد.

Fitness value += 1/string's length

۳-۲-۵ بخش پنجم: تابع بازترکیبی(*crossover function*)

تابع باز ترکیبی یا *crossover function* ، تابعی است که دو کاندید مساله را به جهت تولید کاندید بهتر ، با هم ترکیب می کند و دو کاندید جدیدتولید می کند. همانطور که در بخش ۲-۲-۳ ذکر شد ، گره شماره ۱۳ به عبارتی مرکز ثقل برنامه است. بنا براین تصمیم گرفتیم که دو کاندید انتخاب شده به عنوان والد ، از این نود حساس با یکدیگر ترکیب شوند. برای درک بهتر به شکل زیر دقت کنید :



شکل ۸-۳ دو رشته از نقطه ۱۳ با یکدیگر عوض می شوند و تشکیل دو رشته جدید می دهند : در مرحله انتساب fitness بررسی میشوند و اگر قابل قبول باشند ، باقی خواهند ماند.

۳-۲-۶ بخش ششم:تابع جهش (*mutation function*)

در بخش ۲-۲-۳ به بررسی ساختار برنامه پرداختیم : در تست نرم افزار به روش *white box* ، ساختار به طور کامل در مقابل ما مشاهده می شود . پس چه بهتر که از این موقعیت بهره برداری شود .مهمترین استفاده ما از ساختار در بخش جهش ژنتیکی اتفاق می افتد .چرا که در مراحل اولیه پژوهش جهش ها را به صورت تصادفی اعمال می کردیم .جهش های تصادفی با عث کند شدن روند برنامه و حتی دور شدن از هدف ما می شد . بنا براین با اعمال جهش های هوشمند با استفاده از ساختار ارائه شده در بخش ۲-۲-۳ ، باعث حذف یک سری ژن ها و اضافه شدن برخی از آنها در رشته مسیر شدیم. به عبارتی چند تابع جهش جداگانه برای تغییر ژن ها اضافه نمودیم.

۳-۲-۷ بخش هفتم :نتیجه

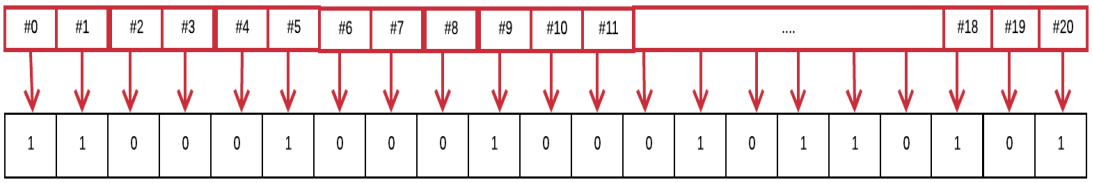
از این برنامه چندین بار اجرا گرفته شد و به طور میانگین در مرحله ، در طی ۱۸ بار تکرار عملیات های ذکر شده ، الگوریتم به مجموعه جواب دست پیدا کرد .

۳-۳ بخش دوم تحقیق : الگوریتم های یادگیری

در بخش دوم از تحقیق ، بنا بر آن شد که مسیر های تصادفی تولید شده در برنامه ، چه درست و چه نادرست را در دیتاستی جمع آوری کرده و تحقیقات آنالیزی را بر روی آن آغاز نماییم. قصد داریم مدلی از روی دیتاست با استفاده از الگوریتم های بحث شده ، بسازیم .هر کدام از این الگوریتم ها را در شناخت مسیر صحیح از مسیر غیر صحیح، عملکرد بهتری از خود نشان دهند را شناسایی کنیم. برای سنجش این امر ، پس از تهیه دیتا ست از کتابخانه *SKlearn*[۲۰] که از تکنولوژی های *machine learning* را در غالب کتابخانه های جداگانه آورده که می توان از توابع آن استفاده کرد.

۳-۳-۱ بخش اول :تعریف ساختار مسیر

در بخش الگوریتم ژنتیک ، ساختاری برای مسیر ها ارائه دادیم که در شکل (شماره)مشهود است. برای به کار گیری الگوریتم های یادگیری ، احتیاج به یکسری ویژگی در مسیر ها داریم که مقدار این ویژگی ها ، برای تصمیم گیری این الگوریتم ها ضروری است . بنابراین ساختار جدیدی را نسبت به الگوریتم ژنتیک ارائه دادیم:



شکل ۹-۳ شماره ها ی هر نود با علامت # مشخص شده است . در صورت طی شدن گره عدد ۱ و در غیر این صورت عدد صفر به خود می گیرد.

در این ساختار تمامی مسیر ها دارای طول یکسان (۲۱) هستند که تعداد تمامی نود های موجود در برنامه است. وجود یا عدم وجود هر یک از نود ها در مسیر تناخاب شده با ۱ و ۰ مشخص می شود . (اگر در مسیر وجود داشته باشد ۱ و اگر وجود نداشته باشد ۰ به آن تعلق میگیرد.) بنا براین این شکل معرف مسیر [0, 1, 5, 9, 13, 15, 16, 18, 20] می باشد که با ساختار جدید نمایش داده شده است.

۳-۳-۲ بخش دوم:ماتریس سردرگمی (*Confusion Matrix*)

به جدول زیر دقت کنید : این جدول برای آنالیز دیتاست های دودویی نقش مهمی را بازی میکند . در واقع به ما نشان می دهد که پس از اطلاق کردن یک الگوریتم به داده های مساله ، تشخیص آن از داده های مساله چگونه بوده است: همانطور که گفته شد ، داده های ما ، مسیر های تصادفی تولید شده و تغییر یافته در الگوریتم ژنتیک می باشند .

[20] www.sklearn.com

این داده ها در یک فایل با فرمت XLS جمع آوری شده اند . به هر کدام از مسیر ها علاوه بر ستون‌های ویژگی ، یک ستون کلاس(class) اختصاص یافته که نشان می‌دهد مسیر تولید شده آیا مسیر صحیحی است یا خیر(کلاس ۰ یعنی مسیر اشتباه و کلاس ۱ یعنی مسیر تولید شده یکی از مسیر های احتمالی برنامه است.)

کار اصلی که میخواهیم با الگوریتم های یادگیری انجام دهیم این آیا که آیا مسیر ها مسیر صحیحی است یا خیر .بنابراین الگوریتم نیاز دارد با مدل که از روی دیتا می سازد، پیش‌بینی کنند که مسیر صحیح است یا خیر .پیش بینی الگوریتم می‌تواند صحیح باشد یا خیر .جدول زیر درک خوبی نسبت به پیش‌بینی های یک الگوریتم به ما می‌دهد .

جدول ۴ ماتریس سردرگمی دیتاست های باینری منبع شکل: *[16]*

		Actual	
		Positive	Negative
Predicted	Positive	**True Positive**	**False Positive**
	Negative	**False Negative**	**True Negative**

در جدول شماره ۴ ، با فرض اینکه کلاس های ۰ و ۱ را کلاس های مثبت (*positive*) و منفی(*negative*) در نظر بگیریم ،جدول بر موارد زیر تاکید می کند:

- TP: چه تعداد از داده های تست را به درستی (*True*) ، درست(*Positive*) پیش بینی شده است. (*True Positive*)
- TN:چه تعداد از داده های تست را به درستی(*True*) ، اشتباه (*Negative*) پیش بینی شده است.(*True Negative*)
- FP: چه تعداد از داده های تست را به اشتباه (*False*) ، درست (*Positive*) پیش بینی شده است. (*False Positive*)
- FN: چه تعداد از داده های تست را به اشتباه(*False*) ، اشتباه(*Negative*)پیش بینی شده است.(*False Negative*)

بنابراین ما نیاز به معیار هایی داریم تا ترکیبی از TP , TN,FP,FN ، میزان دقت پیش بینیمان را بسنجیم .به بررسی مطرح ترین معیار هایی که برای سنجش الگوریتم از آن استفاده می شود ، می پردازیم:

- *Precision*: معیاری که مشخص می کند چه درصدی از آن‌هایی که برایشان کلاس ۱ پیش‌بینی شده است ، واقعاً کلاس ۱ دارند. [16]

$$\text{Precision} = \frac{tp}{tp + fp}$$

- *Recall*: معیاری که مشخص می کند که چه درصدی از آنهایی که واقعا دارای کلاس ۱ هستند را الگوریتم صحیح پیش بینی کرده است. [16]

$$\text{Recall} = \frac{tp}{tp + fn}$$

- *F1-score*: مشکلی که برای خیلی از آنالیز ها ممکن است پیش بیاید این است که کدام معیار را الویت قرار دهیم ؟*precision* یا *recall* ؟؟[16] معیاری که میانگین این دو را محاسبه میکند :

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

بنا بر این ما دو الگوریتم SVM و Decision-Tree را برای دسته بندی دیتاست ، انتخاب کردیم و دقت آن ها را با معیارهایی که در باره آنها توضیح دادیم، خواهیم سنجید.

۳-۳-۳ بخش سوم : دیتاست (*dataset*)

برای ساخت این دیتا ست به یک نکته حائز اهمیت باید توجه داشت:

از آنجایی که ما بیشتر از ۲۰ مسیر صحیح نمی توانیم داشته باشیم ، اگر تعداد رکورد هایی که در دیتا ست قرار می دهیم خیلی زیاد باشد ، حجم زیاد آن را رکورد هایی با کلاس ۰ (کلاس هایی که مسیر درستی نیستند) تشکیل می دهد. بنابراین هنگام تقسیم داده های تست و داده های تمرین ممکن است تعداد زیادی از داده های تست کلاس ۰ داشته باشند و نتیجه درستی را گزارش نکنند.بنابراین تصمیم برآن شد که برای شروع، ۵۰ داده را در فایل بگذاریم که دارای ۲۰ رکورد با "کلاس ۱ "و ۳۰ رکورد با" کلاس ۰" است. داده های فایل ، مسیر های تولید شده توسط الگوریتم ژنتیک است.

۳-۳-٤ بخش چهارم : داده های تمرین و داده های تست (*Train/Test Data*)

در تحقیقات انجام شده در حوزه *machine learning* معمولا بیشتر داده ها را صرف یادگیری و با داده های باقی مانده عملیات تست را انجام می دهند. معمولا نسبت ۳۰ -۷۰ نسبت مناسبی برای داده های تمرین و تست است.یعنی از ۵۰ رکورد ما ، ۳۵ داده برای تمرین و ۱۵ تای باقی مانده برای تست کنار می گذاریم.

نکته حائز اهمیت ، انتخاب تصادفی داده هاست .یعنی لزوما ۷۰ درصد اول برای تمرین و ۳۰ درصد باقی مانده برای تست نباشد . داده های هر دسته بتوانند از هر جایی انتخاب شوند.

۳-۳-۵ بخش سوم :به کار گیری الگوریتم *svm*:

بیشترین استفاده از این متود در مساله های طبقه بندی[21] است .لذا تصمیم گرفتیم مسیر های بوجود آمده در برناممان را طبقه بندی کنیم :آیا مسیر تولید شده مسیر درستی است (کلاس ۱) یا خیر(کلاس ۰). بعد از اینکه با استفاده از SVM مدلی ساختیم ، این مدل را ۲۰ بار با وضعیت های تصادفی متفاوت [22] امتحان کردیم و چند مورد از این نتیجه های پیش بینی را در شکل زیر آورده ایم: وضعیت رندوم متفاوت به این معنا است که هر بار داده های تصادفی عوض می شوند و به همین تبع تعداد کلاس های ۰ و ۱ موجود در داده های تست تغییر می کند.به شکل زیر توجه کنید:

	precision	recall	f1-score	support
0	0.89	0.89	0.89	9
1	0.83	0.83	0.83	6
avg / total	0.87	0.87	0.87	15

	precision	recall	f1-score	support
0	1.00	1.00	1.00	9
1	1.00	1.00	1.00	6
avg / total	1.00	1.00	1.00	15

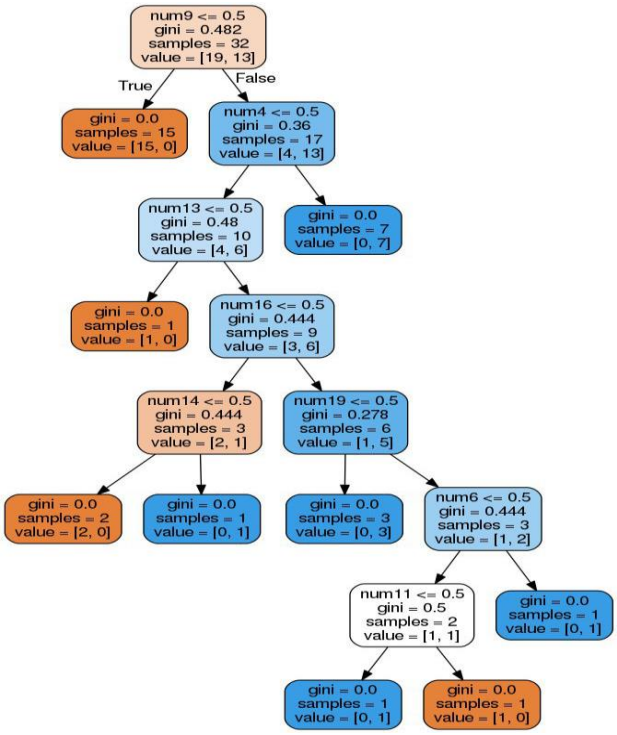
	precision	recall	f1-score	support
0	0.88	1.00	0.93	7
1	1.00	0.88	0.93	8
avg / total	0.94	0.93	0.93	15

شکل۱۰-۳ در این شکل ۳ نتیجه از معیار های های اندازه گیری با داده های مختلف آورده شده است : قابل مشاهده است که در هرکدام از انها مقدار precision و recall و به تبع آن دو f1-score تغییر میکند ولی تغییرات آن چنان مشهود نیست و در راج نسبتا بالایی (نزدیک به ۹۰ درصد) قرار دارد . بنابراین میتوان اذعان داشت الگوریتم تا حد بسیار خوبی درست و به جا استفاده شده است.

[21] *classification*
[22] *Random state*

۶-۳-۳ بخش چهارم : به کار گیری الگوریتم *decision Tree* :

به مانند الگوریتم SVM ، یک مدل از الگوریتم Decision tree میسازیم و با random state های مختلف آن را امتحان می کنیم. هر سطحی از درخت به معنای استخراج یک ویژگی جدید مهم در برنامه است .درنتیجه درخت های تصمیم متفاوتی شکل می گیرد.اما فرقی آنچنانی در گره های اصلی آن مشاهده نمی شود. به چند نمونه دقت کنید:

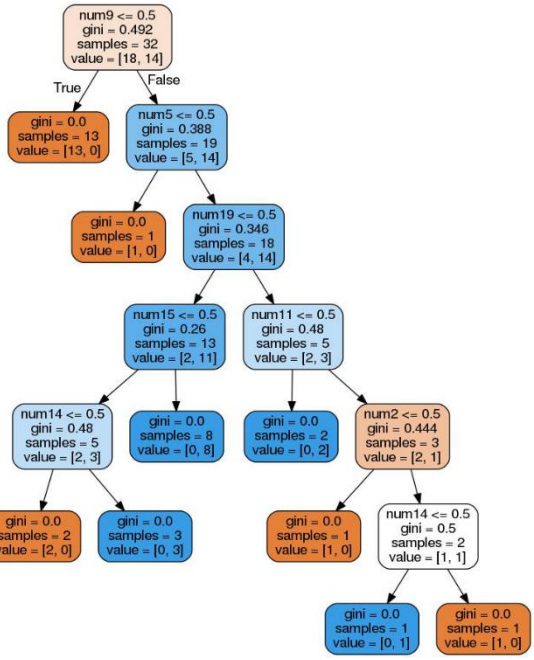


شکل ۱۱-۳ درخت تصمیم

کاهش ۲۰ ویژگی به ۶ ویژگی مهم

ریشه درخت گره شماره ۹

سایر ویژگی های الویت دار به ترتیب عبارت است از :
گره شماره ۵، ۱۹، ۱۵ ، ۱۱ ، ۲، ۱۴



شکل ۱۲-۳ درخت تصمیم

کاهش ۲۰ ویژگی به ۶ ویژگی مهم

ریشه درخت گره شماره ۹

سایر ویژگی های الویت دار به ترتیب عبارت است از :
گره شماره ۵، ۱۹، ۱۵ ، ۱۱ ، ۲، ۱۴

فصل چهارم :نتیجه گیری

الگوریتم ژنتیک با وجود سابقه طولانی اش ، همچنان یکی از مهم ترین الگوریتم های حال حاضر شناخته می شود . تولید اتوماتیک مسیر های برنامه ما با الگوریتم ژنتیک با تعداد محدودی تکرار در مدت زمان قابل قبولی ، با موفقیت رو به رو شد . ***Fitness function*** ارائه شده در عین سادگی ، معیار قابل قبولی برای تست این برنامه به شمار آمد.

الگوریتم های ***SVM*** و ***Decision Tree*** از الگوریتم های معروف طبقه بندی ، نتایج با دقت بالایی به ارمغان آوردند . نکته قابل ملاحظه ، زمان بسیار کوتاه در کشف الگو های دیتاست است. الگو ها و قوانین بدست آمده در نتایج مطابقت نزدیک به ۱۰۰ درصد به الگوهای واقعی داشتند. اما کمبود داده ها به علت کوتاه بودن قطعه کد ، ممکن است در نتایج بسیار خوب این الگوریتم ها بی تاثیر نباشد .

در آینده هدف بر این است که در قسمت الگوریتم ژنتیک ، ***fitness function*** ارتقا داده شود تا نتایج را بهتری حاصل شود. در قسمت الگوریتم های طبقه بندی ، قصد بر آن است که تمرکز بیشتر بر تست اتوماتیک در حوزه ***black box*** در کد های طولانی تر مورد آزمایش قرار گیرد.

منابع

[۱] R. B. a. J. T. Eugenia Díaz, “ercim news,” july 2004 .[درون خطي] .Available: https://www.ercim.eu/publication/Ercim_News/enw58/diaz_e.html.

[۲] P. McMinn, “Search-based Software Test Data Generation: A Survey ”,*Software Testing, Verification and Reliability* .۲۰۰۴ ,۵۸ جلد ,

[۳] D. R. S. Sharmila Jeyarani, “FULLY AUTOMATED TEST CASE GENERATION IN SOFTWARE TESTING USING HILL CLIMBING ALGORITHM ”, *International Journal of Computer Engineering and Applications* جلد ,XII .

[۴] Y. W. ,. Z. Kun Wang, “Software Testing Method Based on Improved Simulated Annealing Algorithm .”

[۵] S. K. R. Yeresime Suresh, “A Genetic Algorithm based Approach for Test Data Generationin Basis Path Testing .۲۰۱۳ ,۷ جلد ”,

[۶] “researchgate,” sep 2014 .[درون خطي] .Available: https://www.researchgate.net/figure/Pseudocode-of-genetic-algorithm_fig6_266746548.

[۷] “researchgate .[درون خطي] ”,Available: https://www.researchgate.net/figure/Pseudocode-of-genetic-algorithm_fig6_266746548.

[۸] M. S. J. V. Boris Delibašić, “White-BoxorBlack-BoxDecisionTreeAlgorithms: Which to Use in Education ”,?*IEEE* .

[۹] “wikipedia .[درون خطي] ”,Available: https://en.wikipedia.org/wiki/Control_flow_graph.

[۱۰] “Evolution of machine learning .[درون خطي] ”,Available: https://www.sas.com/en_ca/insights/analytics/machine-learning.html.

[۱۱] S. E. L. f. A. Testing, “medium .[درون خطي] ”,Available: https://medium.com/@sarahelson81/machine-learning-for-automation-testing-698230917082.

[۱۲] “sklearn .[درون خطي] ”,Available: http://scikit-learn.org/stable/supervised_learning.html#supervised-learning.

[۱۳] S. RAY, “Understanding Support Vector Machine algorithm from examples (along with code),” 13 sep 2017 .[درون خطي] .Available: https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code./

[۱۴] M. S. (. M.), “Chapter 4: Decision Trees Algorithms,” 6 oct 2017 .[درون خطي] . Available: https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1.

[١٥] “Decision Tree - Classification .[درون خطي] ”,Available: https://www.saedsayad.com/decision_tree.htm.

[١٦] A. Jain, “A brief journey on Precision and Recall .[درون خطي] ”,Available: https://towardsdatascience.com/a-brief-journey-on-precision-and-recall-a2651ba99ac6.

[١٧] “tutorialspoint .[درون خطي] ”,Available: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm.

[١٨] “slide player .[درون خطي] ”,Available: https://slideplayer.com/slide/2353642./

[١٩] “Simulated Annealing (SA) .[درون خطي] ”,Available: http://www.lia.disi.unibo.it/Staff/MicheleLombardi/or-tools-doc/user_manual/manual/metaheuristics/SA.html.

[٢٠] A. Q. Fernando Gómez, “Genetic algorithms for feature selection in Data Analytics .[درون خطي] ”,Available: https://www.neuraldesigner.com/blog/genetic_algorithms_for_feature_selection.

[٢١] .[درون خطي]Available: http://www.wardsystems.com/manuals/genehunter/mutation.