پروژه ی پایانی درس رایانش تکاملی

نام : حسین سیم چی

۹۸۴۴۳۱۱۹

استاد : آقای دکتر حامد ملک

۱۳۹۹ / ۱۱ / ۱۷

مقدمه

مسئله ی فروشنده ی دوره گرد به عنوان یک مسئله ی بهینه سازی در سال های اخیر معرفی شده است. به دلیل پیچیدگی زمانی بسیار بالای این الگوریتم، رویکردهای متعدد و زیادی سالیان اخیر برای حل این مسئله معرفی شده اند که الگورتیم های تکاملی بالاترین عملکرد را در بین تمامی این روش ها از خود نشان داده اند. یکی از الگوریتم های مطرح شده و بر اساس هوش جمعیتی و برگرفته از طبیعت، الگوریتم کلونی مورچگان است. در این الگوریتم با الهام از خردجمعی مورچه ها در پیدا کردن کوتاه ترین مسیر جهت یافتن غذا استفاده شده است. مورچه ها زمانی که بر روی مسیری حرکت می کنند ماده ای به اسم فرومون را بر روی زمین می ریزند و مورچه های دیگر اگر بخواهند از مسیر های پیموده شده توسط مورچه های قبل از خود، یک مسیر را انتخاب کنند، مسیری را انتخاب می کنند که دارای مقدار فرومون بیشتری است. و شدت این مقدار را با استفاده از حس بویایی تشخیص می دهند. درنتیجه هرچه مقدار فرومون بیشتر باشد، بوی بیشتری نیز حس خواهد شد. *استفاده از الگوریتم کلونی مورچه ها برای یافتن کوتاه ترین مسیر در مسئله ی فروشنده ی دوره گرد دارای عیوبی است که از جمله ی آن می توان به جستجو نشدن کامل فضای جستجو و زمان نسبت بالای حل آن اشاره کرد. رویکردی که در این پروژه اتخاذ شده است باعث می شود تا زمان نصف شده و همچنین مسیرهای بیشتری جستجو شود تا درنهایت به مقدار مینیمم نهایی نزدیکتر شویم.*

در راستای انجام و بهبود روش ارائه شده، در انتها با تغییر الگوریتم سعی می کنیم تا عملکرد الگوریتم را بهبود دهیم. الگوریتم معرفی شده در مقاله دارای عیوبی است که در روش مطرح شده سعی میکنیم این عیوب را برطرف نموده و روش بهینه تری را ارئه دهیم.

پارامترهای مورد نیاز و توابع نوشته شده

۱. نیاز داریم در ابتدا لیستی تعریف کنیم تا شهرهای خورد را به صورت عدد صحیح در آن ذخیره کنیم که در این پروژه لیست با نام "Ct" قرار داده شده است. این لیست شامل تعداد شهرهای موجود در دیتاست است. به عنوان مثال اگر ۲۰ شهر مختلف داشته باشیم، لیست Ct شامل اعداد یک تا بیست خواهد بود. نمایش مسئله ی فروشنده ی دوره گرد به صورت گراف کامل می باشد و هر راس گراف نمایانگر شهر مورد نظر و یال ها بیانگر میزن فاصله ی بین شهرها است.

۲. تعریف لیست "dist" : در گام بعد باید لیستی تعریف کنیم تا میزان فاصله ی بین شهر ها (رئوس گراف) را در خود نگهداری کند. به عنوان مثال اگر گراف ما دارای ۴ راس (۴ شهر) باشد، لیست dist نهایی ما به صورت زیر خواهد بود و طبق قرارداد هر مقدار را معادل زیر درنظر می گیریم.

$$[[1,1,1,1],[1,1,1,1],[1,1,1,1],[1,1,1,1]]$$

فاصله ی شهر یک به یک (خودش)

فاصله ی شهر دو به سه

فواصل درنظر گرفته شده در لیست بالا همگی برابر یک هستند و برای تشریح استفاده شده اند و واقعی نیستند. برای نشان دادن لیست نام برده شده، از لیست های تو درتو استفاده کرده ایم. در داخل لیست به تعداد شهرها لیست داریم که در هر لیست میزان فاصله ی اون شهر با شهرهای دیگر قرار داده شده اند. ذکر این نکته ضروری است که در این مسئله فواصل هر شهر به خودش تاثیری در حل مسئله ندارند.

۳. تعریف لیست "ph" : دقیقا مشابه مرحله ی قبل در این مرحله نیز لیستی مشابه لیست dist تعریف می کنیم که مقدار فرومون موجود بین هر دو شهر را نشان می دهد. قبل از شروع اجرای الگوریتم باید مطابق نکات مطرح شده در مقاله، میزان فرومون بین هر دو شهر ثابت و برابر با یک درنظر بگیریم.

۴. تعریف تابع g_r_path : از تابع نوشته شده در تابع دیگری به اسم run() استفاده خواهیم نمود. وظیفه ی این تابع، تولید مسیر های تصادفی به طول نصف تعداد شهرها است (مطابق الگوریتم مقاله).

۵. تعریف تابع run1() : وظیفه این تابع این است که مسیر هایی که هر مورچه از راس خود به اندازه ی نیمی از مسیر را تولید کرده ،دریافت کند و سپس با ادغام دو به دو و رندوم این مسیرها، بررسی کند

که آیا با ادغام دو به دو مسیرها یک مسیر از راس آغازی به راس پایانی تولید می شود یا خیر. و سپس به عنوان خروجی، لیستی که تمامی مسیرهای ممکن را پوشش داده است برگرداند. در این الگوریتم فرض شده است که اگر دراثر ادغام لیست ها، تعداد مسیرهای تولید شده بیش از تعداد خاصی باشد مسئله متوقف می شود و به شاخه ی بعدی الگوریتم خواهیم رفت و درغیر این صورت، هر مورچه ادامه ی جستجوی خود را انجام خواهد داد تا مسیر خود را تولید کند.

۶. تابع Road : این تابع وظیفه ی به روز رسانی مقادیر فرومون در لیست فرومون را برعهده دارد. کافی است به عنوان ورودی لیست فرومون، لیست فواصل و مسیرهارا بدهیم. به ازای هر مسیر انتخاب شده مقادیر فرومون در هر مسیر به روزرسانی خواهد شد. خروجی این تابع نیز لیست فرومون به روز رسانی شده است. با استفاده از فرمول زیر مقدار فرومون هر یال را به روزرسانی خواهیم کرد

$$phromone\ (t+1) = phromone\ (t) \times Evapuration + \frac{phromone}{distance}$$

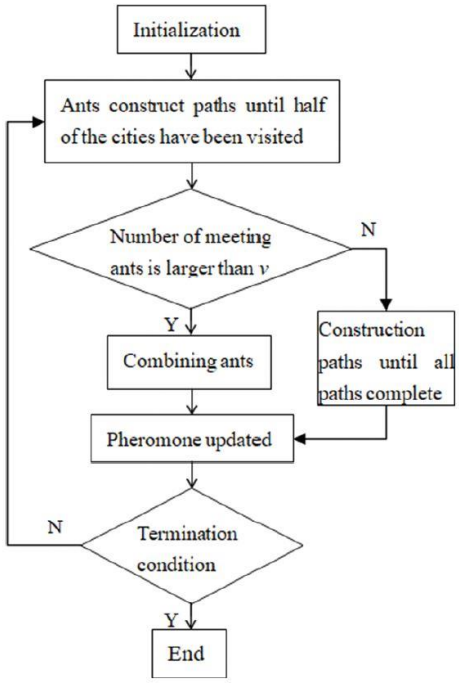
یکی از مشکلات الگوریتم پیشنهادی مقاله این است که به مقدار تبخیر (Evapuration) بسیار وابسته است. اگر مقدار تبخیر را بزرگ قرار دهیم، در تعداد مراحل تکرار بالا ممکن است مقادیر فرمون ها به بینهایت نزدیک شود که درنتیجه انتخاب را برای مورچه های بعدی سخت می کند. اگر مقدار تبخیر را نیز خیلی کوچک قرار دهیم، در تعداد تکرار بالا ممکن است مقدار فرومون یال ها به صفر همگرا شود. خب طبیعتا انتخاب مقدار تبخیر توسط کاربر صورت می گیرد که مشکل بسیار بزرگی خواهد بود، درنتیجه سعی می کنیم با استفاده از رویکرد جدید این مشکل را برطرف نماییم. از فرمول فوق قابل برداشت است زمانی که فاصله ی بین دو راس از حدی بیشتر باشد، مقدار اضافه شده خیلی کوچک خواهد بود. لذا تاثیر کوتاه بودن یا بلند مسیر در به روزرسانی فرومون ها نقش بسیار مهمی خواهد داشت.

۷. به ازای تعداد تکرار خاص، توابع run() و run1() را اجرا می کنیم تا شانس جستجوی مناطق دیده نشده افزایش یابد.

۸. تابع Ph : بر اساس لیست نهایی فرومون ها، از هر راس مسیری را انتخاب می کنیم که اولا مینیمم مسسیر ممکن پیدا شده توسط الگوریتم باشد و ثانیا مسیری باشد که دارای بیشترین مقدار فرومون را داشته باشد. (البته با توجه به الگوریتم موجود درمقاله، مانند گام هفتم، باید نیمی از مسیر تولید شود و شروط بیان شده چک شود)

۹. تابع Costs : زمانی که مسیرهای نهایی ما تولید شوند، با استفاده از این تابع و لیست dist، مقدار هزینه ی نهایی مسیر انتخاب شده به عنوان عدد نهایی الگوریتم برگشت داده می شود.

مراحل گفته شده بر اساس الگوریتم نوشته شده در مقاله نوشته و پیاده سازی شده اند که در زیر فلوچارت کلی آن را مشاهده می کنید



مشکلات الگوریتم فوق

۱. وابسته بودن نتیجه ی الگوریتم به مقدار تبخیر که توسط کاربر مشخص می شود.
۲. تعداد جواب های تولید شده به مقدار تبخیر بستگی دارد. درصورتی که مقدار تبخیر به درستی انتخاب نشود ممکن است تعداد جواب نسبتا خوبی تولید نشود. به عنوان مثال فرض کنید اگر مسئله ی ما دارای ۱۰ جواب نزدیک به مینیمم سراسری باشد (۱۰ مسیر نزدیک به حداقل مقدار هزینه)، در این حالت اگر مقدار فرومون را به درستی انتخاب نکنیم ممکن است تنها ۴ مسیر از ۱۰ مسیر ممکن بدست بیاید

مزایای الگوریتم

۱. سرعت بالا تسبت به الگوریتم های ماقبل خود.
۲. پایداری بالای الگوریتم بدین معنا که جواب های نهایی همگی نزدیک به مینیمم سراسری یا برابر با آن هستند

نتایج و تحلیل بدست آمده از کد الگوریتم موجود در مقاله

جهت ارزیابی کد نوشته شده باید در ابتدا تعداد شهرها و میزان فاصله ی بین آن ها را به الگوریتم داده تا بتواند با توجه به مراحل بیان شده، کوتاه ترین مسیر نزدیک به بهینه ی سراسری را به ما برگرداند.

برای شروع مانند تمامی مقالات چاپ شده در زمینه ی TSP، میخواهیم با تعداد راس های کوچک و میزان وزن های مشخص کار را شروع کنیم و سپس مقدار بازگردانده شده از الگوریتم را با مقدار نهایی خود مقایسه کنیم در نتیجه از ۳ دیتاست قدیمی استفاده خوهیم کرد

برای حالات زیر لیست dist به صورت زیر تعریف شده است (۲ دیتاست اول) :

$$\underbrace{[[1,2,\dots,n],[1,2,\dots,n],\dots,[1,2,\dots,n]]}_{\mathbf{n}}$$

که در لیست فوق n برابر با تعداد شهر های انتخاب شده می باشد

۱. **دیتاست اول**، با تعداد شهرهای برابر ۴ الگوریتم را اجرا می کنیم. دقت داشته باشید زمانی که تعداد راس و به تبع آن تعداد یال ها کم باشد باید تعداد مراحل خاصی را برای اجرای الگوریتم یا به اصطلاح شرط خاتمه انتخاب نماییم زیرا زمانی که تعداد مراحل از حدی بیشتر شود، مقدار فرومون روی یال ها به سمت بینهایت همگرا خواهد شد که البته سرعت همگرایی آن وابسته به مقدار تبخیر است. اگر مقدار تبخیر را خیلی کوچک انتخاب کنیم با سرعت بسیار زیادی مقادیر فرومون ها به سمت صفر همگرا خواهد شد.

- مینیمم فاصله در این حالت برابر مقدار ۱۰ میباشد که الگوریتم در این حالت با شرط خاتمه یا تکرار ۳ بار و مقدار تبخیر برابر با ۰.۵ به عدد ۱۰ همگرا خواهد شد

مقدار نهایی فرومون پس از اتمام الگوریتم به صورت زیر خواهد بود

```
[[0, 1.0, 1.1362994285727442e-106, 3.2730170493037632e-31],
 [1.2073434749826929e+235, 0, 1.6120653277377767e-28, 3.2399425046315175e-22],
 [16100687809804.729, 1.0, 0, 6.891945124656732e-181],
 [7.74255068301112e+78, 1.0, 4.1530715945566046e-14, 0]]
```

به لیست مقادیر فرومون ها دقت کنید، مقدار فرومون هر راس به خودش به روزرسانی نشده است و مقدار صفر دارد، زیرا در این الگوریتم و مسئله حرکتی از هر راس یا شهر به خودش را نداریم. با کمی دقت نیز می توان فهمید مسیر هایی که طول کمتری دارند دارا مقدار فرومون بیشتری نیز هستند پس در گام بعد انتخاب خواهند شد.

```
[[1, 2, 3, 4], [2, 1, 3, 4], [3, 1, 4, 2]]
[10, 10, 10]
```

همچنین مطابق لیست فوق، در انتها مسیر های انتخاب شده به صورت بالا می باشند و مقدار هزینه های آن نیز نوشته شده است. به عنوان مثال مسیر اول برابر با لیست اول می باشد یعنی ابتدا از راس یک به دو سپس از دو به سه در ادامه از سه به چهار و دوباره در انتها به راس اول برمی گردیم. که هزینه ی آن نیز برابر با عدد ۱۰ می باشد. در واقع لیست مسیرهای فوق نشان می دهد از سه راس یک، دو و سه می توانیم مقدار مینیمم سراسری برسیم.

۲. **دیتاست دوم**، تعداد راس ها یا شهرها را برابر با ۶ درنظر می گیریم. در این حالت بر اساس نکات گفته شده در دیتاست ها، مقدار بهینه ی سراسری برابر با ۲۱ می باشد که با شرط توقف برابر ۵ به این عدد

همگرا خواهیم شد و در انتها با شروع از هر شهر درصورتی که شروط گفته شده را دارا باشد مسیر و هزینه ی آن را برمی گرداند.

```
[[0, 1.0, 6.833440307307545e-39, 5.891378415370064e-42, 2.5925410561971236e-30,
1.4494889247084975e-75], [8.067519707095138e+97, 0, 1, 3.424148199040195e-71,
2.7617006512385884e-26, 1.1720749684213083e-114], [6.267782277194411e+148, 1.0,
0, 4.504120532389345e-59, 2.7122745293527606e-61, 1], [4.414056233274322e+55,
1.0, 3.0232619255547523e-16, 0, 1.8312421332353887e-131, 1.350397829426094e-8
8], [2.592321479487944e+26, 1.0, 1, 2.5221569128804044e-55, 0, 2.69808408929906
72e-64], [279210559319.2102, 1.0, 5.08999617854669e-100, 1.8122132210517458e-1
7, 1.5081052258653771e-53, 0]]

[[2, 1, 3, 4, 5, 6], [3, 1, 4, 2, 6, 5], [4, 1, 3, 2, 6, 5], [5, 1, 3, 2, 4,
6]]
[21, 21, 21, 21]
```

این الگوریتم توانسته ۴ مسیر را پیدا کند که مارا به بهینه ی سراسری می رساند که برابر با کمترین هزینه ی ممکن برای مسئله ی فروشنده ی دوره گرد است.

***** شرط خاتمه، مقدار متغیر تبخیر و تعداد Ants meeting ها به عنوان چالش اصلی این الگوریتم می باشند که در صورتی که به درستی انتخاب نشوند مسئله از جواب درست دور خواهد شد یا اصلا به جواب نخواهد رسید.**

۳. **دیتاست سوم**، دیتاست Five

مقدار مینیم دیتاست عدد ۱۹ است و خروجی نهایی الگوریتم ما به صورت زیر خواهد بود (مقادیر پارامترها به این صورت تنظیم شده اند که مقدار تبخیر برابر ۰.۵ و شرط خاتمه برابر ۳ درنظر گرفته شده است)

```
[[1, 4, 5, 2, 3], [4, 1, 3, 5, 2]]
[19, 23]
```

FIVE is a set of 5 cities. The minimal tour has length 19.

همانطور که قابل مشاهده است به خوبی توانسته ایم مقدار مینیمم مسیر را برای این دیتاست بدست آوریم. به لحاظ پایداری نیز الگوریتم کاملا پایدار است زیراکه جواب دوم بدست آمده نیز نزدیک به بهترین جواب بدست آمده است

*** وجود متغیر تبخیر و شرط خاتمه ی الگوریتم نمی تواند به صورت سراسری انتخاب شود. **که خود دلیلی بر معتبر نبودن روش معرفی شده است**. در تمامی منابع موجود در اینترنت تعداد بسیار زیادی دیتاست وجود دارند که همگی آن ها دارای تعداد شهر و فواصل معین و مخصوص به خود هستند. درنتیجه تعیین مقدار مشخص برای پارامترها به ازای تمام دیتاست ها کار درستی نیست و در بعضی اوقات ما را به جواب نهایی نمی رساند. البته در مقاله نیز ذکر شده است که بر اثر تجربه و عملیات بر روی دیتاست های استفاده شده مقدار بهینه ی پارامترها را مشخص کرده ایم که این نیز دلیل خیلی خوبی برای ثابت درنظر گرفتن پارامترها نمی تواند باشد. دلیل بد بودن فرض ثابت درنظر گرفتن مقادیر پارامترها این است که شما فرض کنید ۵ شهر دارید و مقادیر فاصله ی بین این شهرها بین بین عدد ۱ تا ۲۰ متغیر باشد و از طرفی در دیتاست دیگر ۲۰ شهر داریم و عدد فاصله ی بین شهر ها بین ۳۰۰ تا ۶۰۰ متغیر باشد. در این صورت آیا با تعداد شرط خاتمه ی یکسان به جواب خواهیم رسید ؟؟؟!!!! پاسخ به وضوح منفی است چراکه در دیتاست دوم به دلیل افزایش تعداد شهرها نیاز داریم تا به صورت رندوم جستجوی بیشتری نیز انجام دهیم. و در دیتاست اول برعکس. این ایراد را می توانیم با درنظر گرفتن شرط خاص برطرف نماییم ولی بازهم میزان تبخیر بسیار موثر خواهد بود. چرا که می دانیم هرچه مقدار متغیر تبخیر کمتر باشد سرعت تبخیر بیشتر خواهد بود و درنتیجه زمانی که فواصل بین یال ها در دیتاستی خاص بسیار کوچک باشند به سرعت در تعداد مراحل کم به صفر همگرا خواهند شد. در این صورت به نظر بنده به همین خاطر در بیشتر رفرنس های مربوط به بیشینه ی مقاله ی منتشر شده مقدار متغیر تبخیر را برابر ۱ درنظر می گیرند. نکات و راه حل های بیان شده جهت رفع مشکلات در کد اعمال شده و نتایج به صورت زیر خواهد بود.

جمع بندی مشکلات موجود در الگوریتم از دیدگاه بنده

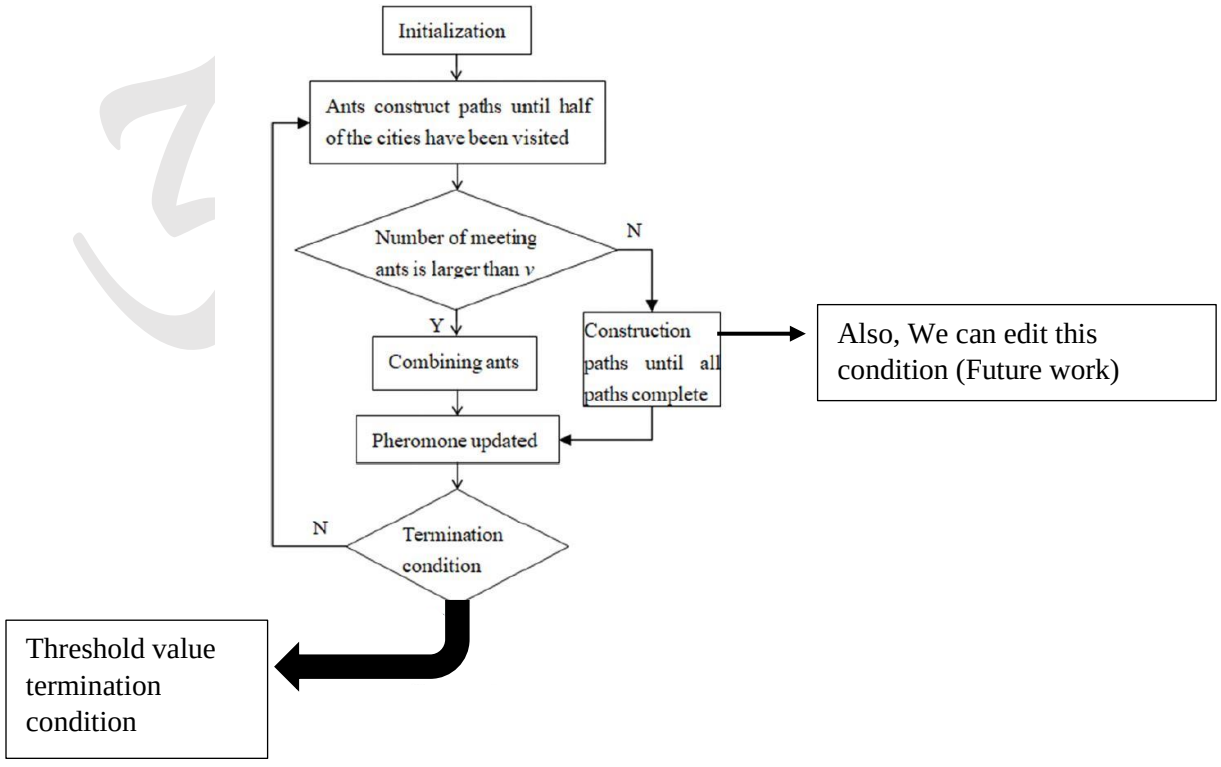
۱. الگوریتم موجود بر روی یکسری از دیتاست ها به خوبی عمل کرده (۳ مورد اول بررسی شده) درحالیکه این تعداد ممکن است در بین انواع بسیار زیاد دیتاست ها بسیار کم باشد. البته ممکن است دیتاست های دیگری که استفاده کرده ام استاندارد نبوده باشند و به اشتباه مقدار مینیمم نهایی را قرار داده باشند
۲. شرط گذاشته شده (Meeting strategies) مشخص نبوده و در اکثر مواقع رعایت نمی شود
۳. شرط خاتمه بر اساس توضیحات داده شده نمی تواند در تمام دیتاست ها یکسان باشد
۴. مقدار متغیر تبخیر نمی تواند در تمامی دیتاست ها یکسان باشد

راه حل های بررسی شده جهت بهبود عملکرد

۱. برای رفع مشکل دوم می توان تعیین کرد تا زمانی که شرط گذاشته شده رعایت نشده است الگوریتم به تولید مسیرهای تصادفی ادامه دهد (متاسفانه به دلیل کمبود وقت در این پروژه بررسی نشده است)
۲. برای رفع مشکل سوم نیز می توان مقادیر را از بین بازه ای مشخص انتخاب نمود که به صورت Adaptive بر روی الگوریتم اعمال شود. این خاصیت کمک می کند تا مقدار مسیر بهینه ی خورجی الگوریتم به طور چشم گیری بهبود یابد. درنتیجه برای اعمال این تغییر از مقدار استانه استفاده می کنیم، مقدار استانه مقداری است که اگر از آن مقدار کمتر به الگوریتم مقدار دهیم الگوریتم قادر به یافتن جواب بهینه نخواهد بود. درواقع مقدار استانه به صورت Adaptive و براساس دیتاست مشخص خواهد شد درنتیجه مقدار استانه به الگوریتم ما کمک خواهد کرد تا جواب های بدست آمده از الگوریتم پایه را بهبود دهیم. با تکنیک مشخص کردن مقدار آستانه دیگر نیازی به تنظیم کردن مقدار شرط خاتمه نخواهیم داشت.
۳. برای بهبود مشکل چهارم نیز مقدار متغیر تبخیر را برابر ۱ یا ۰.۹ درنظر می گیریم.

درنتیجه با راه های بررسی شده در بالا می توانیم عملکرد الگوریتم را بهبود دهیم که در این صورت خاصیت تعمیم پذیری الگوریتم نیز افزایش خواهد یافت. بدین معنا که در اکثر دیتاســت های موجود جواب خوبی را ارائه کند. تکنیک Adaptive بودن به افزایش خاصیت تعمیم پذیری الگوریتم کمک خواهد کرد

درنتیجه الگوریتم بهبود یافته به صورت زیر تغییر پیدا خواهد کرد



حال به ادامه ی نتایج بدست آمده بر روی سه دیتاست معروف دیگر نیز می پردازیم

۴. **دیتاست چهارم،** FRI26

در این دیتاست، ۲۶ شهر با فواصل معین از یکدیگر قرارگرفته اند و نتایج کد بهبود یافته به صورت زیر می باشد

```
FRI26 is a set of 26 cities
Enter your value:(Start with 2500) 2500
[2386, 2264, 2493, 2287, 1869, 2284, 2251, 2349, 2286, 2291, 2288, 2131, 2379,
2470, 2263, 2270, 2268, 2074, 2191, 2263]
```

مقدار آستانه در این دیتاست بر اساس مقدار بهینه ی دیتاست و تکرارهای زیاد عدد ۲۵۰۰ بدست آمده است. مقدار مینیمم فاصله عدد ۹۳۷ می باشد که الگوریتم بهبود یافته عددی نزدیک به ۱۸۶۹ را بدست آورده است.

قابل ذکر است به دلیل رندوم بودن انتخاب در مسیرهای اولیه در هربار جواب مینیمم متفاوت خواهد بود.

حال اگر الگوریتم اولیه را بر روی دیتاست لحاظ کنیم خواهیم داشت :

```
FRI26 On First algorithm
[2496, 2518, 2653, 2652, 2338, 2522, 2770, 2781, 2700, 2682, 2528, 2641, 2735,
2747, 2648, 2594, 2835, 2579, 3143, 2913, 2830, 2429]
```

همانطور که از بالا مشخص است جواب بهینه ی بدست آمده از الگوریتم اولیه ۲۳۳۸ است که اختلاف زیادی با الگوریتم توسعه یافته دارد.

همچنین با مقایسه ی خروجی های دو الگوریتم خواهیم فهمید که الگوریتم توسعه یافته به لحاظ پایداری وضعیت بهتری نسبت به الگوریتم اولیه دارد و این هم بخاطر وجود مقدار آستانه است که تمامی جواب ها از مقدار آستانه کمتر می باشند.

۵. **دیتاست پنجم،** GR17

در این دیتاست، ۱۷ شهر با فواصل معین از یکدیگر قرارگرفته اند و نتایج کد بهبود یافته به صورت زیر می باشد

```
GR17 is a set of 17 cities
Enter your value:(Start with 3500) 3500
[3383, 3415, 3274, 3467, 3431, 3431, 3465, 3465, 3465, 3465, 3465, 3319, 3319,
3319, 3319, 3319, 3319, 3319, 3319, 3319]
```

مقدار آستانه در این دیتاست بر اساس مقدار بهینه ی دیتاست و تکرارهای زیاد عدد ۳۵۰۰ بدست آمده است.

مقدار مینیمم فاصله عدد ۲۰۸۵ می باشد که الگوریتم بهبود یافته عددی نزدیک به ۳۲۷۴ را بدست آورده است.

و نتایج مربوط به الگوریتم اولیه

```
GR17 On First algorithm
[4638, 5509, 5069, 4402, 5116, 4264, 4579, 4385, 4672, 4851, 4234, 4650, 5069,
4468, 4567, 4671]
```

بقیه ی تحلیل ها مانند قسمت قبل کاملا قابل بیان و نتیجه گیری است. مقدار مینیمم بدست آمده توسط این الگوریتم نیز ۴۲۳۴ است.

۶. **دیتاست ششم،** P01

در این دیتاست، ۱۵ شهر با فواصل معین از یکدیگر قرارگرفته اند و نتایج کد بهبود یافته به صورت زیر می باشد

```
P01 is a set of 15 cities
Enter your value:(Start with 500) 550
[505, 544, 535, 522, 534, 534, 494, 494, 530, 530, 540, 540, 548, 548, 548, 54
8, 548, 548, 548, 548]
```

مقدار آستانه در این دیتاست بر اساس مقدار بهینه ی دیتاست و تکرارهای زیاد عدد ۳۵۰۰ بدست آمده است.

مقدار مینیمم فاصله عدد ۲۹۱ می باشد که الگوریتم بهبود یافته عددی نزدیک به ۴۹۴ را بدست آورده است.

و نتایج مربوط به الگوریتم اولیه

```
P01 On First algorithm
[682, 695, 711, 734, 645, 695, 716, 668, 690, 608, 633, 705, 633]
```

نتیجه گیری پایانی و مقایسه دو الگوریتم بیان شده

هنوز هم نتایج بدست آمده بر روی برخی دیتاست ها خوب نیست و تفاوت و فاصله ی زیادی با مسیرهای بهنیه دارند. **ولی در این پروژه سعی کردیم ایرادات الگوریتم یاد شده را شناسایی کرده و راه حل هایی نیز جهت بهبود آن ارائه دهیم**. در الگوریتم توسعه یافته وابستگی شدید الگوریتم به متغیرها را از بین بردیم و تلاش کردیم تعداد متغییر هایی که باید مقدار دهی کنیم از عدد ۴ به ۱ برسانیم. همچنین قابلیت تعمیم الگوریتم نیز افزایش یافت و توانستیم نشان دهیم که اگر مقدار استانه ی خوبی را انتخاب کنیم، الگوریتم می تواند جواب های قابل قبولی اطراف بهینه ی سراسری به ما برگرداند. همچنین بر این باور دارم که بخش Meeting strategy موجود در الگوریتم در اکثر مواقع رعایت نشده و الگوریتم از این مرحله عبور خواهد کرد. درنتیجه اگر مکانیزمی ارائه دهیم که بتواند حتما این بخش را پوشش دهد قطعا نتایج نزدیکتری بر روی برخی دیتاست ها خواهیم گرفت.

البته الگوریتم معرفی شده همانطور که در اوایل بحث نیز به ان اشاره شد بر روی برخی از دیتاست ها به خوبی عمل میکند، مانند ۳ دیتاست کوچکی که در ابتدا به آن ها پرداختیم که دقیقا مقدار بهینه ی سراسری را پیدا کرده است. ولی به طور کلی باتوجه به بحث های صورت گرفته و دانش اینجانب در زمینه ی عملی سازی این مقاله، قابلیت تعمیم بر روی تمامی دیتاست ها را نداشته که خود قابل تامل است.

در روش ارائه سعی شد : (روش توسعه یافته)

۱. کاهش تعداد متغیر هایی که باید مقدار دهی شوند از ۴ به ۱
۲. افزایش پایداری جواب های مسئله
۳. افزایش قابلیت تعمیم
۴. نزدیک شدن بیشتر به مقدار سراسری در مقایسه با الگوریتم موجود در مقاله

باتشکر، حسین سیم چی، ۹۸۴۴۳۱۱۹

۱۳۹۹ / ۱۱ / ۱۷