به نام خدا

دانشگاه صنعتی شریف

دانشکده مهندسی برق

هوش مصنوعی و محاسبات زیستی ـ دکتر حاجی پور

نیم‌سال اول ۱۳۹۹

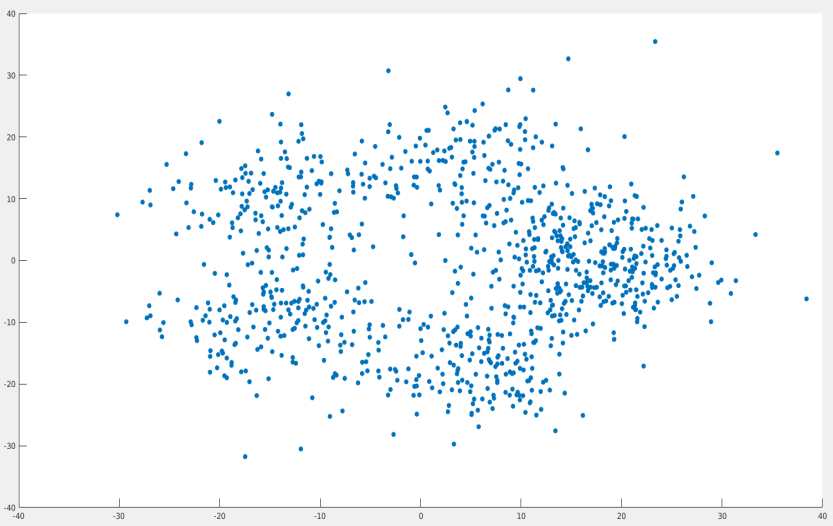
گزارش بخش کامپیوتری تمرین سری چهارم (امتیازی)

امید شرفی (۹۶۱۰۱۸۳۸)

فهرست مطالب

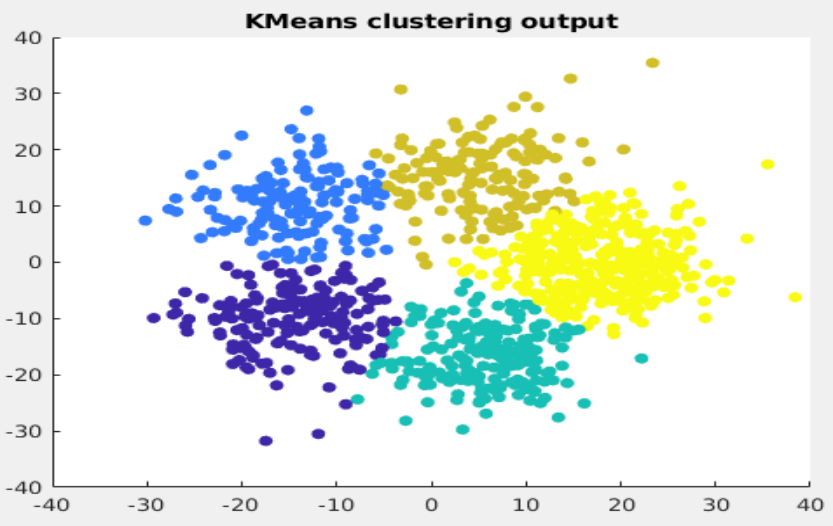
۱ مقدمه

داده های ورودی مساله شامل مختصات هزار نقطه در فضای دو بعدی هست. داده ها در پنج دسته میباشند.



شکل ۱: نمایش داده ها

همچنین خروجی مساله با روش k-means به صورت زیر است.



شکل ۲: خوشه بنده k-means

حال در ادامه به حل همین مساله با الگوریتم های دیگر خواهیم پرداخت.

۲ الگوریتم ژنتیک

۱.۲ نحوه پیاده سازی

- نحوه ی رمزگذاری : برای رمزگذاری مساله میتوانیم یک کروموزوم به طول ۱۰۰۰ در نظر بگیریم که هر آلل نماینده ی یک نفطه هست و ژن ها اعداد ۱ تا ۵ هستند که نماینده ی دسته ی هز نقطه میباشد.
- تابع هزینه : برای حالتی که حداکثر ۵ مرکز داریم که چون مختصات هر نقطه و دسته ی آن نقطه را برای پاسخ داریم برای هر دسته جدا مرکز دسته را از روی نقاط آن دسته حساب میکنیم و سپس جمع نرم۲ فواصل را هر نقطه با مرکز های هر نقطه را به عنوان تابع هزینه میگیریم. همچنین در حالتی که دقیقا ۵ پاسخ مدنظر هست یک شرط اضافه ی تعداد دسته های موجود در پاسخ را هم بررسی میکنیم که اگر زیر ۵ دسته باشد یک ترم خیلی بزرگ با هزینه جمع میکنیم که پاسخ به عنوان پاسخ مطلوب انتخاب نشود.
- جمعیت اولیه : برای جمعیت اولیه میتوان به صورت رندوم انتخاب کرد. برای مساله ی دقیقا ۵ دسته بهتر هست که در جمعیت اولیه حتما هر ۵ دسته با تعداد قابل قبولی نقطه برای هر دسته به عنوان شروع کار انتخاب شود که ما در این بخش برای جمعیت اولیه از ۵ دسته ی ۲۰۰ نقطه ای به صورت رندوم به عنوان جمعیت اولیه انتخاب کردیم.
- جهش و ترکیب : برای ترکیب میتوان از ترکیب چند نقطه ای و برای جهش نیز از جهش رندوم استفاده کرد.

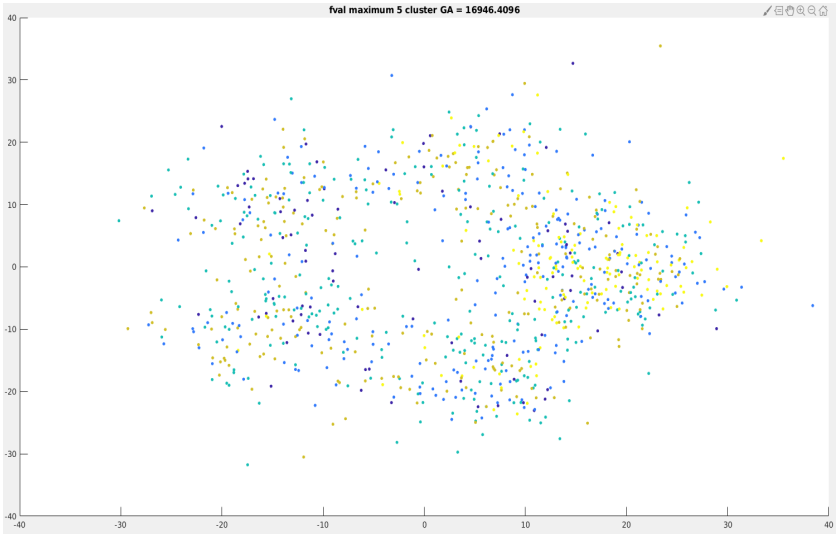
۲.۲ بررسی کد

در نتیجه برای پیاده سازی از تولباکس ga استفاده شد. همچنین تنظیامات و وضعیت نسل اولیه به فرمت زیر تعیین شد.

```
% first method (not good result)
init_vector = [ones(1,200), 2.*ones(1,200), 3.*ones(1,200), 4.*ones(1,200), 5.*ones(1,200)];
init_vector = init_vector(randperm(1000));
options = optimoptions(options,'InitialPopulationMatrix', init_vector);
options = optimoptions(options,'FitnessScalingFcn', @fitscalingprop);
options = optimoptions(options,'SelectionFcn', @selectionroulette);
options = optimoptions(options,'CrossoverFcn', @crossovertwopoint);
options = optimoptions(options,'Display', 'off');
options = optimoptions(options,'PlotFcn', {  @gaplotbestf @gaplotscorediversity @gaplotscores });
```

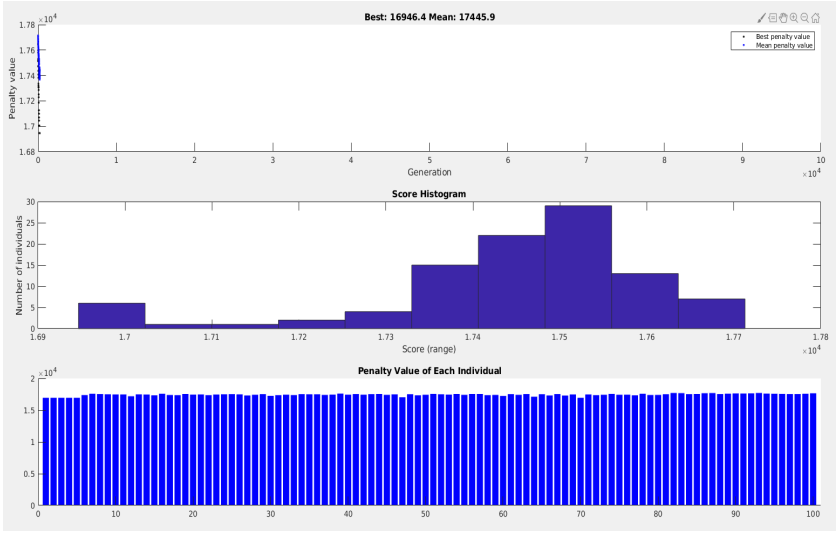
شکل ۳: تنظیمات متد اولیه ی الگوریتم ژنتیک

کد با این روش پیاده سازی شد. پس از اجراهای متعدد با تنظیمات توابع مختلف همانطور که در شکل پایین هم دیده میشود نتیجه مطلوب با این رمزگذاری حاصل نشد. در حقیقت اتفاقی که رخ میدهد سرعت تغییرات در روش به حدی نیست که بتواند سریع و خوب به نتایج خوب همگرا شود.



شکل ۴: روش رمزگذاری اول به جواب مطلوب همگرا نمیشود.

در نتیجه همانطور که در زیر مشاهده میشود نتیجه هیچ کدام از ژن ها در روش اول در یک نسل مطلوب نیست.



شکل ۵: روش رمزگذاری اول به جواب مطلوب همگرا نمیشود.

برای همین در ادامه روش رمز گذاری را عوض کردیم:

- نحوه ی رمزگذاری : روموزوم را به طول ۱۰ گرفتیم که هر جفت آلل نماینده یک مرکز در فضا هست.
- تابع هزینه : برای محاسبه ی تابع هزینه برای حالت حداکثر ۵ مرکز متناسب با هر مرکز خوشه که در کروموزوم مشخص هست حساب میکنیم که هر داده به کدام یک از این دسته ها نزدیک تر هست و در

نتیجه با استفاده از مجموع فواصل هزینه محاسبه میشود. برای حالت دقیقا ۵ هم یک تابع هزینه ی بالا مشابها در نظر میگیرم.

- جمعیت اولیه : در مورد چمعیت اولیه محدودیت خاصی وجود ندارد فقط شرط بازه ی مرکز ها وجود دارد. در نتیجه چون داده های مساله در هر بعد بین منفی چهل تا چهل بود این محدودیت را در شروط ga اعمال میکنیم.
- جهش و ترکیب : چون اکنون مساله پیوسته شده است برای ترکیب میتوان از درونیابی و برای جهش از تابع گوسی استفاده میکنیم.

تنظیمات در روش دوم به صورت زیر هست :

```
options = optimoptions('ga');
options = optimoptions(options,'FitnessScalingFcn', @fitscalingprop);
options = optimoptions(options,'SelectionFcn', @selectionroulette);
options = optimoptions(options,'CrossoverFcn', @crossoverarithmetic);
options = optimoptions(options,'MutationFcn', {  @mutationgaussian 1  0.7 });
options = optimoptions(options,'Display', 'off');
options = optimoptions(options,'PlotFcn', {  @gaplotbestf @gaplotscorediversity @gaplotscores });
```

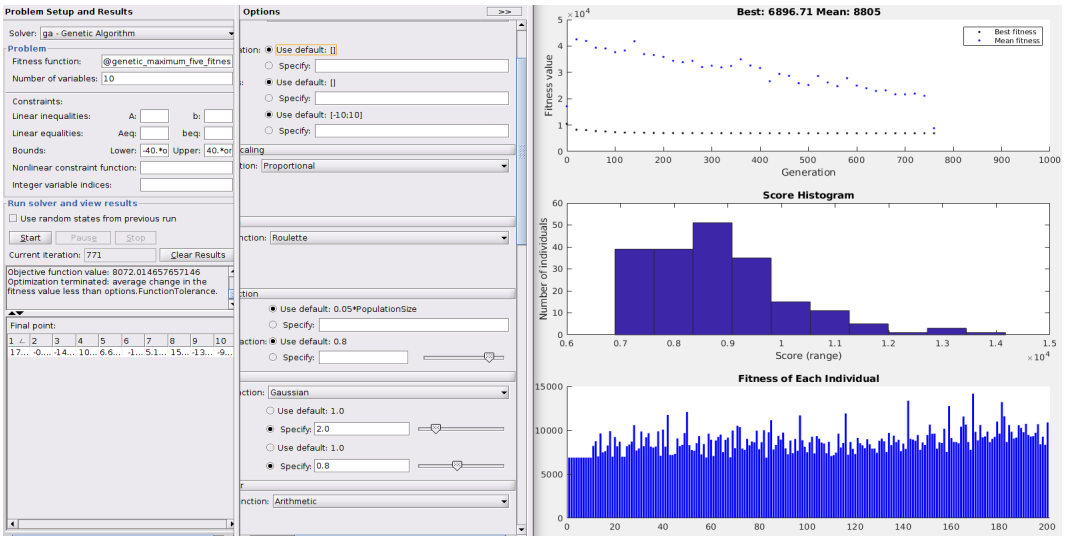
شکل ۶: تنظیمات رویکرد دوم الگوریتم ژنتیک

یکی از فاکتور های مهم واریانس و ضریب انقباظ تابع گوسی جهش هست. فرمول این ضریب به صورت زیر است:

$$\sigma_k = \sigma_{k-1}\left(1 - \text{Shrink}\frac{k}{\text{Generations}}\right)$$

شکل ۷: رابطه ضریب انقباض و واریانس تابع گوسی جهش الگوریتم

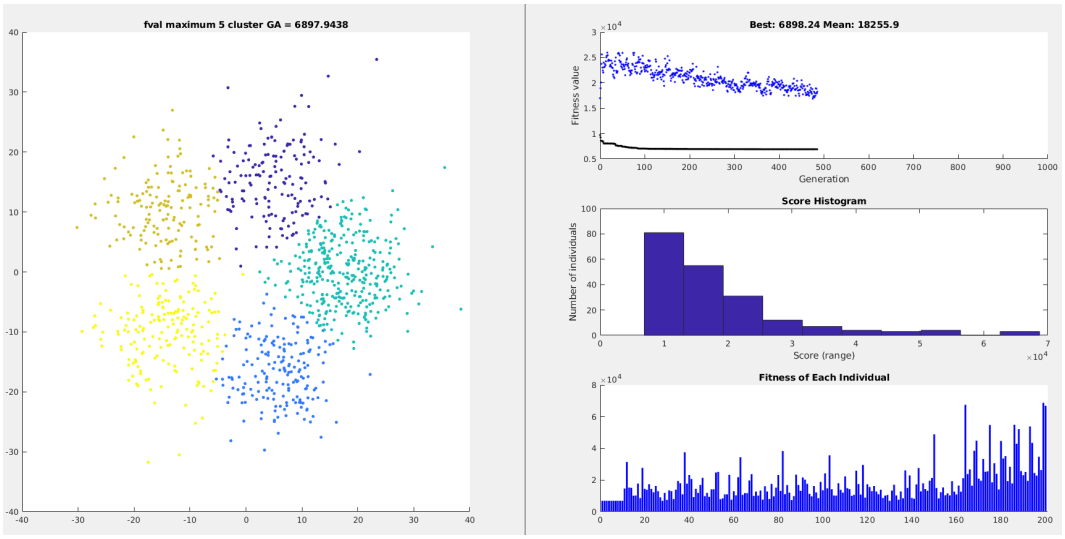
و وقتی حالت های مختلف تست شد از سمتی واریانس اولیه اگر خیلی کوچک باشد مساله ممکن است خیلی خوب نگردد و به پاسخ های خوب نرسیم. از طرفی اگر خیلی هم بزرگ باشد ممکن است دقت مطلوبی که در نهایت کار احتیاج داریم و تغییرات جزئی که در انتهای کار احتیاج داریم موجود نباشد. در نتیجه انگار وقتی واریانس اولیه زیاد باشد مساله سریع میتواند فضا را بگردد و به نتایج خوب برسد اما برای این که دقت بهینگی پاسخ بهینه را بالاتر ببریم باید واریانس در آن مرحله کم باشد. از طرفی در مورد انقباض واریانس نیز به همین صورت. اگر خیلی سریع منقبض کنیم ممکن است خیلی طول بکشد که پاسخ ها فضا را بگردند و اگر هم خیلی طول دهیم که کم شود ممکن است مساله خیلی طول بکشد. در زیر یک مثال از این موضوع با ضریب انقباض بالا که ۷۷۱ مرحله برای همگرایی طول کشیده است و نهایتا هم نتیجه حاصل خیلی مطلوب نیست.



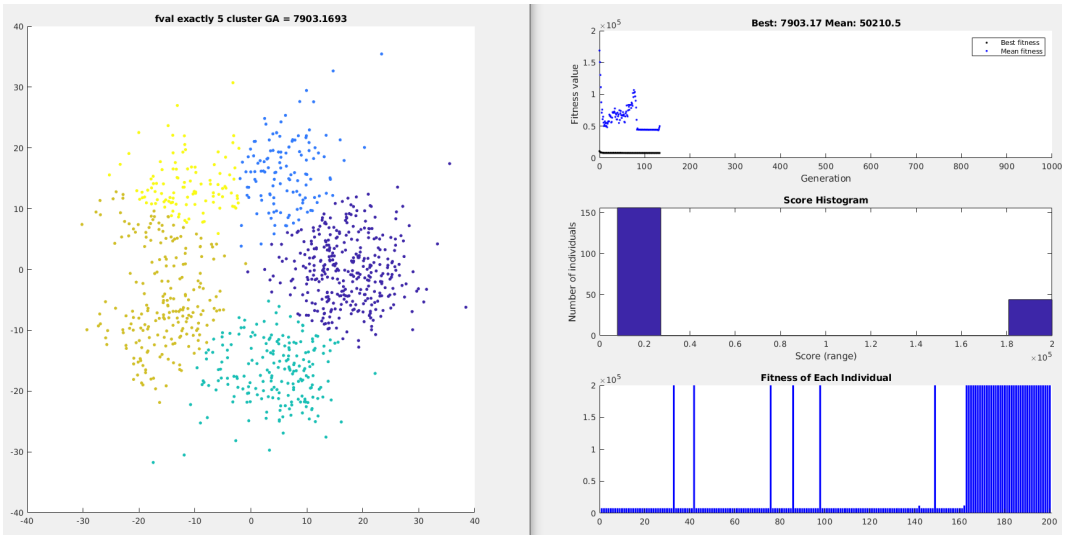
شکل ۸: ضریب انقباض نامناسب برای تابع جهش گوسی

۳.۲ نتایج

در ادامه نتایج رویکرد دومی که معرفی کردیم برای دو حالت حداکثر پنج دسته و دقیقا پنج دسته آورده شده است:



شکل ۹: نتیجه با تابع هزینه حداکثر پنج دسته



شکل ۱۰: نتیجه با تابع هزینه دقیقا پنج دسته

نهایتا در انتهای این بخش دو نکته جالب است. یک این که همانطور که در ژن های یک نسل در حالت دقیقا پنج دسته مشاهده میشود تعدادی پاسخ با هزینه خیلی بالا وجود دارد که در حقیقت همان پاسخ هایی هستند که کمتر از پنج دسته دارند و در نتیجه جریمه شده اند.

یک موضوع دیگری هم که مطرح هست این است که خب مشخص است که پاسخ بهینه پنج خوشه خواهد داشت یعنی حالا که مساله مشخصا میتواند خوب به پنج دسته تقسیم شود و در نتیجه نتیجه ی بخش حداکثر پنج دسته همان دسته را دارد. ولیکن فارغ از این موضوع حتی اگر مثلا مساله به فرمتی باشد که دو دسته هم خوب تقسیم شود مشخصا میتونیم یک داده رو بدیم به یک دسته خالی و هزینه کم کم میشود اما میدانیم که یک دسته با یک داده عملا یک دسته به مفهوم خوشه ای که مدنظر ما هست نیست.

در نتیجه اگر هدفمان پیدا کردن تعداد دسته های خوب برای حل مساله در کنار خوشه بندی هست میتوانیم تابع هزینه الگوریتم ژنتیک را به این صورت تغییر دهیم که اولا برای حالت دقیق وقتی که تعداد دسته ها را داریم بگوییم مثلا اگر زیر یک درصدی از کل داده ها به یک خوشه افتاد جریمه ی زیادی برای هزینه آن پاسخ قرار دهیم. در عین حال برای حالتی هم که دوست داریم که مساله خودش تعداد خوشه ها را به دست آورد (مشخصا میدانیم که در روش خوشه بندی نمودار هزینه ی مجموع فواصل داده ها با مرکز هر خوشه نسبت به هایپرپارامتر تعداد خوشه به یک زانو دارد که تعداد خوشه های مطلوب ما همان نقطه ای هست که نمودار زانو میزند وگرنه که همواره هزینه بر حسب تعداد خوشه ها نزولی هست) بیایم و برای دسته ها با تعداد داده های خیلی کم متناسب با عکس تعداد داده های آن خوشه جریمه روی تابع هزینه اعمال کنیم که در نتیجه تعریف خوشه یا به تعریف مطلوب و منطقی ما که حداقل یک تعداد خوب داده داشته باشد نزدیک تر شود یا الکی خوشه با تعداد خیلی کم داده ایجاد نشود.

۳ الگوریتم ازدحام ذرات

۱.۳ نحوه پیاده سازی و بررسی کد

در الگوریتم ازدحام ذرات مشابه روش کدگذاری بخش قبل هر ذره به طول ده نماینده مختصات پنج مرکز خواهد بود و تابع هزینه نیز برای حالت دقیقا پنج دسته یک هزینه جریمه زیاد خواهد داشت تا پاسخ از حالت مطلوب دور شود.

برای پیاده سازی الگوریتم از دستور particleswarm متلب استفاده میکنیم و شروط منفی چهل تا چهل برای مختصات مرکز در هر بعد اعمال میکنیم. نهایتا هم برای نمایش متناسب با پاسخی که ذره بهینه دارد نزدیک ترین مرکز را برای هر داده محاسبه و نقاط را با رنگ مربوط به هر دسته ترسیم میکنیم.

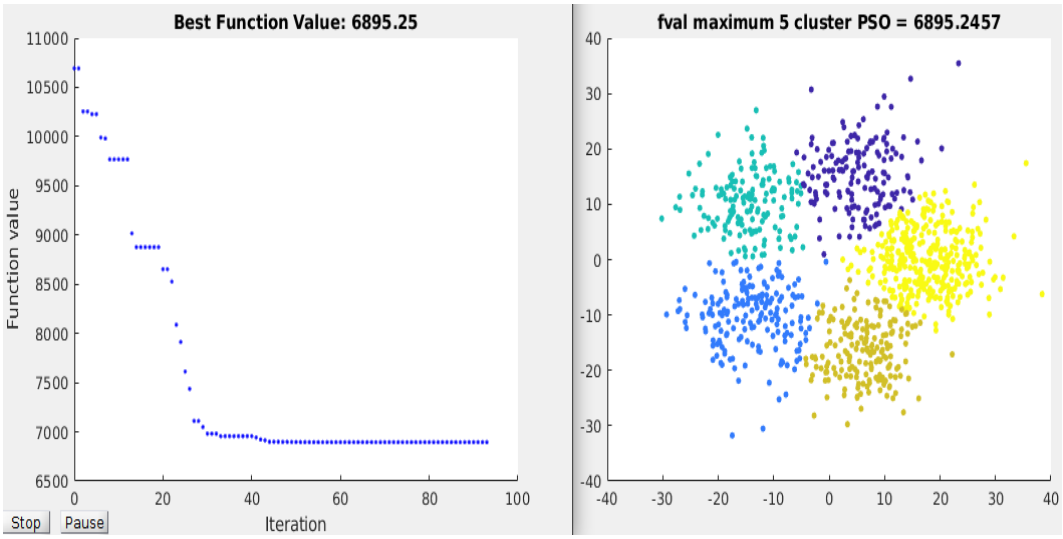
```
%% maximum 5 clusters
options = optimoptions('particleswarm');
options = optimoptions(options,'PlotFcn', {  @pswplotbestf });
[result, fval] = particleswarm(@genetic_maximum_five_fitness_1, 10, -40.*ones(1,10), 40.*ones(1,10), options);

% Show result
c = ones(1,1000);
for i = 1:1000
    minn = norm([result(1), result(2)] - [x(1,i), x(2,i)]);
    for j = 2:5
        if norm([result(2*j-1), result(2*j)] - [x(1,i), x(2,i)]) < minn
            minn = norm([result(2*j-1), result(2*j)] - [x(1,i), x(2,i)]);
            c(i) = j;
        end
    end
end
figure()
scatter(x(1,:),x(2,:),15,c, 'filled');
title(['fval maximum 5 cluster PSO = ' num2str(fval)]);
```

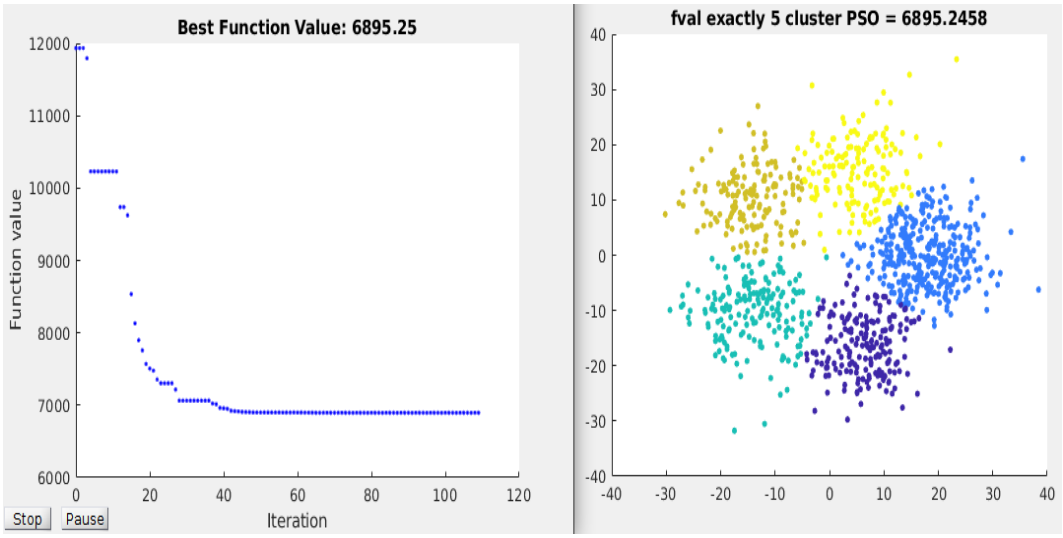
شکل ۱۱: پیاده سازی الگوریتم ازدحام ذرات

۲.۳ نتایج

نتایج الگوریتم به صورت زیر است. در حالت دقیقا پنج دسته جون یک سری از پاسخ ها که زیر پنج دسته هستن هزینه ی زیادی دارند میانگین در اول بالاتر از حالت دیگر است و کمی هم تعداد تکرارهای مورد نیاز تا پاسخ همگرا شود کمی بالاتر میرود.



شکل ۱۲: نتیجه با تابع هزینه حداکثر پنج دسته



شکل ۱۳: نتیجه با تابع هزینه دقیقا پنج دسته

۴ الگوریتم بهینه سازی مورچگان

۱.۴ نحوه پیاده سازی

برای حل مساله ی دسته بندی الگوریتم دو رویکرد میتوان پیش رو گرفت:

۱.۱.۴ رویکرد اول

در رویکرد اول ما یک گراف دو بخشی تشکیل میدهیم. در یک سمت به تعداد دسته ها که در اینجا پنج هست گره قرار میدهیم که نماینده هر دسته هست و در سمت راست به تعداد داده ها که اینجا هزار هست داده قرار میدهیم. در ادامه از این گراف دو بخشی کامل باید یک زیر گراف انتخاب کنیم که درجه ی هر راس سمت داده ها دقیقا یک باشد. همچنین از سمت دسته های نیز اگر بخوهیم دقیقا پنج دسته داشته باشیم درجه ی هر راس باید حداقل یک باشد. البته اگر نکته ای که در الگوریتم ژنتیک گفتیم را رعایت کنیم باید این حداقل درجه از یک عددی متناسب با حجم نقاط داده ها بالاتر باشد تا مفهموم دسته را داشته باشیم.

در ادامه هزینه پاسخ را محاسبه میکنیم و هر چقدر که کمتر باشد، فرمون بیشتری روی یال های این پاسخ در گراف دو بخشی قرار میدهیم. مثلا معکوس مجموع فواصل رو فرومون گذاری میکنیم. همچنین در انتها نیز اگر بخواهیم دقیقا پنج دسته باشیم در مخرج کسر با مجموع فواصل یک عدد بزرگ جمع میکنیم که فرومون خیلی کمی قرار گیرد و آن پاسخ تقویت نشود.

همچنین از دیگر کارهایی که میتوان کرد اولا متناسب با تعداد اعضای اون دسته یا وضعیت مرکز فعلی هر راس با راس ما را در انتخاب خود به جز فرومون به عنوان اطلاعات اضافه دخیل کنیم. همچنین بحث ترتیب مجدد را نیز به این صورت در انتهای فرومون گذاری قرار دهیم که نام دسته ای که بیشترین تکرار را دارد ۱ قرار دهیم و به همین صورت نام گذاری دسته ها را متناسب با تعداد اعضا انجام دهیم که پخش شدگی فرومون کمی کمتر شود.

۲.۱.۴ رویکرد دوم

در رویکرد دوم نیز میتوانیم یک گراف کامل تشکیل دهیم و سپس متناسب با فرومون بین داده ها یال قرار دهیم تا جایی که به k دسته ی مورد نظر برسیم، سپس هر زیر گراف را کامل کنیم و سپس متناسب با هزینه مشابه رویکرد قبل فرومون گذاری را انجام میدهیم. این روش هم میتواند خوب باشد ولیکن از دید اوردر حل مساله با توجه به این که بعد از قرار دادن هر یال باید چک کنیم که گراف چند زیر گراف همبند دارد در نتییجه اولا تعداد یال ها از اودر nk که در مساله قبل بود به توان دوم n رسیده است و همچنین چون باید هر سری تعداد زیر بخش های همبند گراف را هم چک کنیم که از اوردر رمانی n هست و مجموعا مساله برای هر مرحله از یک فرم خطی بر حسب n به یک فرم توان سوم از n خواهد رسید. به طور مثال عملا همین الان که ۱۰۰۰ داده داریم این روش عملا برای یک مورچه از اوردر ده به توان نه یعنی در حدود حداقل یک ثانیه طول خواهد کشید که در نتیجه اگر یک کلونی با صد مورچه را هزار مرحله تکرار کنیم حدود ۳۰ ساعت اجرای کد طول خواهد کشید که اصلا منطقی نمیباشد.

۲.۴ بررسی کد

برای پیاده سازی کد اولا یک مدل برای گراف نیاز داریم که:

$$fG(i,j) = pheromone(i,j)$$

همچنین یک ضریب تبخیر و بودن یا نبودن این ضریب، مقدار اولیه فرومون های روی یال های گراف، دو ضریب آلفا و بتا برای میزان تاثیرگذاری فرومون قبلی و اطلاعات اضافه ای که ذکر شد در آپدیت فرومون مرحله بعد از هایپرپارامتر های مدل خواهد بود.

```
for ant=1:nAnt
    s = 0; % sum of
    for j = 1:1000
        s = sum(fG(:,j));
        r = s.*rand();
        p = 0;
        for k = 1:5
            p = p + fG(k,j);
            if( r <= p )
                c(ant, j) = k;
                break;
            end
        end
    end

    % Calculate ant's answer cost and update fG
    cost = genetic_maximum_five_fitness(c(ant, :));
    costs(ant) = cost;

end

for ant=1:nAnt
    fG = fG .* fEvapapiration; % Evapapiration
    f = 100000 ./ (costs(ant)^3);

    % Update fromone
    for j = 1:1000
        fG(c(ant, j), j) = fG(c(ant, j), j) + f;
    end
end
```

شکل ۱۴: پیاده سازی الگوریتم مورچگان

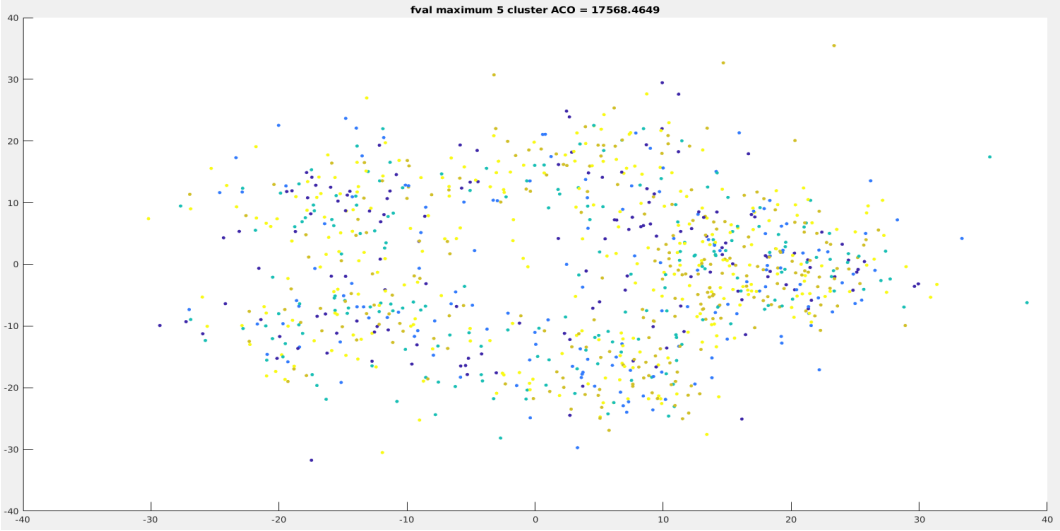
همچنین خود فرمول آپدیت نیز قابل تنظیم هست و شاید یکی از مهم ترین بخش ها نیز باشد. به طور مثال شما میتونید معکوس هزینه رو به عنوان فرومونی که اضافه میشود قرار بدید. میتونید معکوس هزینه به توان سه رو قرار بدید در نتیجه هر چه هرینه بیشتر شود میزان افزایش فرومون افت شدیدتری پیدا میکند. میتوان هر ترکیب خطی از هزینه را قرار داد و به همین ترتیب که یک سری از حالت های مختلف آزمایش شد.

۳.۴ نتایج

در اولین تلاش برای مساله صرفا یک مورچه در نظر گرفتم و حلقه را تکرار کردم. در ادامه از صد مورچه استفاده کردم و توابع مختلف هزینه را بررسی کردم. از یک طرف از فرم های درجه های بالا یعنی آلفای بالا برای بیشتر کردن تاثیر مجموع فواصل استفاده کردم. همچنین چون ما اورودر مجموع را میدانیم که از پنج

هزار بالاتر است برای بیشتر کردن این تاثیر نیز از حذف این بایاس هم اضافه کردم و همچنین ترم فاصله و تعداد دسته را به عنوان اطلاعات اضافه تست کردم.

همچنین یک تابع reorder نوشتم که متناسب با تعداد اعضای هر دسته مرتب کرده و لیبل گذاری را مجدد اعمال میکند که در نتیجه بخواهد پخش شدگی فرومون را کم کند و در نهایت یک کد دسته بندی با الگوریتم مورچگان از روی github نیز تست شد که به نظر میرسد در مجموع این مساله حداقل با این تنظیماتی که تست شد خیلی برای مساله ی kmeans مانند مساله ی فروشنده دورگرد که کاملا روی مدل مورچگان میشیند خوب عمل نمیکند و پخش شدگی و پیچیدگی مساله مخصوصا برای این دیتاست که خیلی هم مشخص دسته ها مجزا نیستند پاسخ خوبی نمیدهد و بیشتر به سمت رندوم سرچ الگوریتم رفته است. نمونه ای از پاسخ که مناسب نیز برای این بخش در ادامه آورده شده است.



شکل ۱۵: نتیجه الگوریتم مورچگان با آلفای ۳ و مرتب سازی در نام گذاری. با تبخیر نه دهم و فرومون اولیه ی یک روی یال ها. اوردر فرمونی که پاسخ بهینه قرار میدهد در حدود ۳۰ و برای دسته بندی رندوم حدود ۲ میباشد.

۵ مقایسه سه الگوریتم

با توجه به نتایجی که دیده شد در وحله ی اول به نظر میرسد الگوریتم مورچگان تا حدودی روی این مساله مانند مساله ی فروشنده دورگرد خوب نمیشیند و جواب مطلوبی حاصل نمیکند چون ذات مساله به صورتی هست که باعث میشد هرچقدر هم که مرتب سازی پاسخ و اصلاح انجام دهیم بایز هم بیشتر به سمت رندوم سرچ به علت پخش شدگر فرومون میرود و حداقل با این مدل که هزار راس در بخش راست گراف در نظر میگیرم شاید مدل مطلوبی نباشید. البته ممکن است با اعمال شدید اطلاعات خارجی و انتخاب فرمول ویژه ای برای بخش آپدیت کردن به نتایج بهتری برسیم.

اما میان دو روش دیگر یعنی ژنتیک و ازدحام ذرات که هردو پاسخ مطلوبی دادند از دید پیچیدگی پیاده سازی خب پیاده سازی الگوریتم ژنتیک با توجه به مراحل و پیچیدگی هایی که دارد از روش ازدحام ذرات

سخت تر خواهد بود. در ازدحام ذرات الگوریتم سرراست تر هست که البته ما درگیر پیاده سازی ها نشدیم و تولباکس آماده موجود بود. در عین حال همین پیچیدگی نشان میدهد که هر تکرار و تولید نسل بعد در الگوریتم ژنتیک بیشتر از الگوریتم ازدحام ذرات زمان میبرد و در نتیجه همانطور که در عمل هم دیده شد الگوریتم ازدحام ذرات سریع تر به پاسخ مطلوب رسید. نهایتا اما یک مزیت خیلی خوب الگوریتم ژنتیک امکانات و گزینه های مختلف هست. به این صورت که گستره ی بزرگتری از مسائل را میتوان با توابع و تنظیمات مختلف در مراحل مختلف الگوریتم اعمال کرد و در نتیجه در مسائل در حین حال که الگوریتم ذرات هم میتواند خیلی سر راست و ساده به نتیجه ی مطلوب برسید اما استفاده از الگوریتم ژنتیک نیز برای داشتن توانایی های بیشتر بسیار مفید خواهد بود.