دانشگاه شهید باهنر کرمان

دانشکده فنی و مهندسی

گروه مهندسی کامپیوتر

الگوریتم های هوش جمعی

پیاده سازی الگوریتم ICA در محیط پایتون

استاد ارجمند:

جناب آقای دکتر بحرالعلوم

دانشجویان:

نجمه نظری (۹۹۱۵۶۰۰۶)

امیرحسین حسنی(۹۹۱۵۵۰۰۴)

محمد اسماعیلی (۹۹۱۵۵۰۰۱)

قسمت اول) تعریف پارامتر ها

قسمت دوم) مراحل اجرای الگوریتم

قسمت سوم) گزارش نتایج

تعریف پارامتر ها :

Num_countries : تعداد کل کشور ها و یا کلونی ها

Num_imp : تعداد استعمارگران (imperialists)

Lbound : یا حد پایین که مختصات هر نقطه در هر بعد از آن کمتر نمی شود.

Ubound : یا حد بالا که مختصات هر نقطه در هر بعد از آن بیشتر نمی شود.

Beta : مقداری بزرگتر از ۱ که در فاصله کلونی تا استعمارگر ضرب می شود و در مقدار فاصله ای که کلونی به استعمارگر نزدیک می شود تاثیر دارد. این مقدار طبق مقاله و برای همگرایی بهتر معمولا برابر ۲ می باشد.

Teta : کلونی ها با این زاویه نسبت به استعمارگر حرکت می کنند. این زاویه معمولا بین مقدار pi/4- تا pi/4 انتخاب می شود.

Zeta : مقداری که اگر کم باشد باعث می شود تا قدرت(هزینه) یک استعمارگر توسط خودش تعیین گردد و هر چه کوچکتر باشد تاثیر کلونی ها بر روی قدرت یک استعمارگر بیشتر می شود.

این مقدار کمتر از ۱ و معمولا برابر ۰.۱ می باشد.

Num_of_goals : در صورتی در یک تابع یا مسئله به دنبال چند هدف باشیم می توان به جای ۱ اعداد بیشتری قرار داد. این کار باعث می شود تا الگوریتم تا جایی ادامه دهد که به تعدادی که مشخص کردیم استعمارگر باقی بماند. هر استعمارگر در واقع یک هدف می باشد.این ویژگی به صورت ذاتی در الگوریتم وجود دارد.

algorithem_iteration : تعداد باری است که الگوریتم دوباره از اول اجرا می شود و در نهایت میانگین مقدار تابع به دست آمده از تکرار های مختلف برای هر هدف به صورت جداگانه و با یک لیست نمایش داده می شود.

مراحل اجرای الگوریتم :

۱) مقداردهی اولیه به کلونی ها:

این کار با استفاده از numpy.random.randint و پارامتر هایی که در ابتدای کار تعریف کردیم صورت می پذیرد. همچنین تعداد کلونی هایی که به هر استعمارگر داده می شود با توجه به فرمول های زیر مشخص میگردند.

در کد پیاده سازی کلونی ها به صورتی است که یک لیست ابتدا مقادیر مکان آن ها قرار دارد و سپس مقدار تابع (در اینجا برابر مقدار شایستگی) قرار دارد و در نهایت شماره ای که نشان دهنده تعلق هر کلونی به استعمارگر را نشان می دهد.

```
(x, y, function value, number of imperialist)
```

$$C_n = c_n - \max_i \{c_i\}$$

cn مقدار هزینه استعمارگر nام و Cn هزینه نرمال شده هر استعمارگر می باشد.

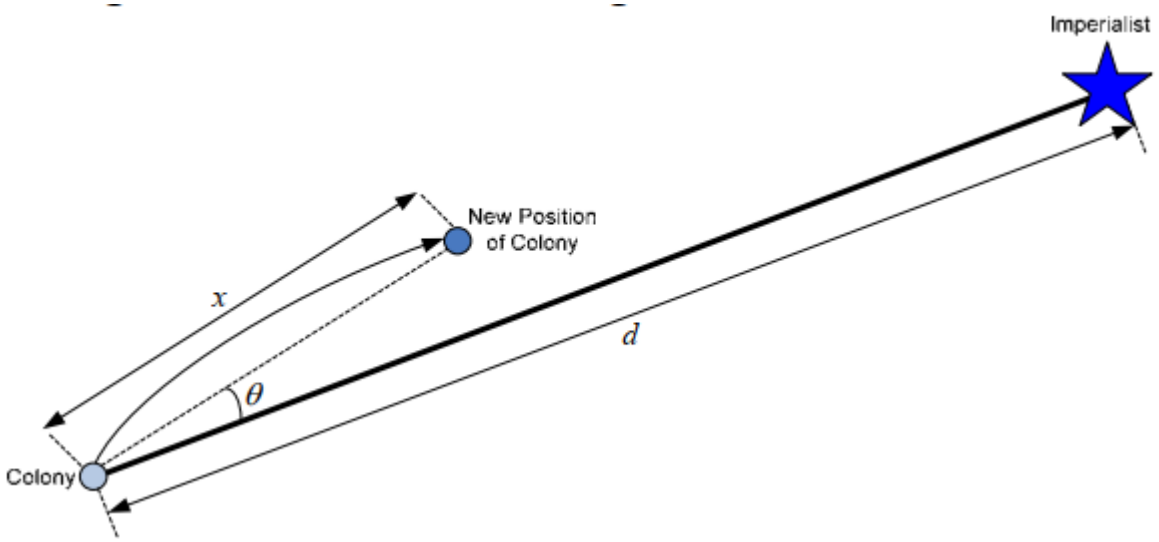
$$p_n = \left| \frac{C_n}{\sum_{i=1}^{N_{imp}} C_i} \right|$$

Pn قدرت نرمال شده هر استعمارگر می باشد.

$$N.C._n = round\{p_n . N_{col}\}$$

NCn تعداد کلونی هایی که به هر استعمارگر تعلق می گیرد می باشد که از گرد شده ی مقدار ضرب Cn و Pn به دست می آید.

۲) نزدیک کردن کلونی ها به استعمارگران:



در اینجا هدف به دست آوردن پارامتر x و اعمال آن به مختصات هر کلونی می باشد. برای انجام این کار در کد از پارامتر هایی که در قسمت ابتدایی تعریف کردیم استفاده می کنیم.

$$x \sim U(0, beta*d)$$

x مقدار بین ۰ تا beta*d در توزیع نرمال به خود می گیرد.

x = numpy.random.uniform(low=0, high=beta*dist)

۳) بررسی کلونی های هر استعمارگر : در صورتی که کلونی با هزینه بهتری وجود داشته باشد در محدوده یک استعمارگر وجود داشته باشد تبدیل به استعمارگر می شود و استعمارگر قبلی تبدیل به یک کلونی.

۴) پیدا کردن ضعیف ترین کلونی از ضعیف ترین امپراتوری و دادن آن به امپراتوری که بیشترین احتمال دریافت کلونی را دارد.

$$T.C._n = Cost(imperialist_n) + \xi\, mean\{Cost(colonies\ of\ empire_n)\}$$

ابتدا تمام هزینه یک امپراتوری(استعمارگر به علاوه ی کلونی هایش) محاسبه می شود.

Cost یا هزینه در واقع مقدار تابع ماست. مقداری که در بخش دوم فرمول در میانگین ضرب شده است را با نام zeta تعریف کردیم و همانظور که گفتیم مشخص میکند که در هزینه کل, هزینه کلونی ها و یا هزینه استعمارگر بیشترین تاثیر را دارد.

$$N.T.C._n = T.C._n - \max_i \{T.C._i\}$$

فرمول بالا هزینه کلی نرمال شده می باشد.

$$p_{p_n} = \left| \frac{N.T.C._n}{\sum_{i=1}^{N_{imp}} N.T.C._i} \right|$$

احتمال مالکیت Ppn می باشد که با فرمول بالا محاسبه می شود و ضعیف ترین کلونی از ضعیف ترین امپراتوری به امپراتوری که بیشترین احتمال مالکیت را داراست تعلق می گیرد.

۵) حذف کردن ضعیف ترین امپراتوری یا امپراتوری که فقط استعمارگر آن باقی مانده است و همه کلونی هایش را از دست داده است.

۶) اگر به تعداد مورد نظر امپراتوری باقی مانده است الگوریتم متوقف می شود و در غیر این صورت به مرحله ۲ می رویم.

گزارش نتایج :

این الگوریتم بر روی تابع چند هدفه G1 که در مقاله اصلی از آن استفاده شده پیاده سازی شده است. علاوه بر آن الگوریتم امکان پیدا کردن اهداف دیگر با تغییر تعداد امپراتوری های باقی مانده را دارد.

num_countries = 100

num_imp = 10

lbound = 0

ubound = 10

beta = 2

teta = np.random.uniform(low=-math.pi/4, high=math.pi/4)

zeta = 0.1

num_of_goals = 2

gorithem_iteration = 10

در عکس های زیر مشاهده میکنیم که در اجرای اول با ۳۰ بار تکرار(۳۰ شروع مجدد الگوریتم) برای هدف اول میانگین ۱۴.۶۳- و برای هدف دوم میانگین ۱۱.۹۸- به دست آمده. در اجرای دوم به ترتیب ۱۵.۴۸- و ۱۲.۳۴- را برای اهداف اول و دوم داشتیم.

مقدار کمینه تابع ۱۸.۵۵- می باشد که الگوریتم ما جواب نسبتا قابل قبولی را به دست آورده است.

```
ICA()
```

```
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  6.           9.         -12.86824392   1.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           8.         -11.45955887   0.        ]
 [  9.           8.         -11.45955887   0.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           5.         -11.91812579   0.        ]
 [  9.           8.         -11.45955887   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   4.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  6.           9.         -12.86824392   0.        ]
 [  9.           8.         -11.45955887   3.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.08838317   2.07781271 -10.79926213   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           5.         -11.91812579   0.        ]
 [  9.           5.         -11.91812579   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   3.        ]]

Mean function value of goal1, goal2, ..., goaln after 10 repeats=  [-14.632875490864492, -11.983387220287215]
```

```
ICA()
```

```
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           5.         -11.91812579   4.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.0178043    8.65688067 -18.51984446   0.        ]
 [  9.           8.         -11.45955887   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  6.           9.         -12.86824392   0.        ]
 [  9.           5.         -11.91812579   3.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  6.           9.         -12.86824392   2.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   1.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  9.           6.         -12.46739094   1.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  6.           9.         -12.86824392   0.        ]
 [  9.07967633   5.07014719 -12.57427032   1.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           6.         -12.46739094   2.        ]
 [  9.           6.         -12.46739094   1.        ]]
winning imperialists =
(x, y, function value, number of imperialist)
 [[  9.           9.         -16.36078342   0.        ]
 [  6.           9.         -12.86824392   2.        ]]

Mean function value of goal1, goal2, ..., goaln after 10 repeats=  [-15.488842377293157, -12.347613235810508]
```