حل مسئله چند هدفه با قید

ستاره روشن

چکیده — در این گزارش به بررسی عملکرد هفت الگوریتم تکاملی بر روی مسئله یافتن ترتیب کودها با ماکزیمم دسترس پذیری همچنین ماکزیمم بهروری به اضافه دو قید خواهیم پرداخت. در این تمرین از دو روش direct و indirect در تامین قیود استفاده کردیم.

۱. پیش پردازش‌ها

در ابتدا دیتاستی را که در task چهارم بدست آورده بودیم فراخوانی می‌کنیم. سپس در آرایه قرار داده و روی اسامی کودها و کشورها factorize انجام می‌دهیم.

```
[4]: 1 df = pd.read_excel(r"C:\Users\mcf\OneDrive\Desktop\Availability_ Brazil.xlsx")
     2 samples = np.asarray(df)

[5]: 1 df
```

Out[5]:

	country	Fertilizers	Import&Product	Export	Availability	Use	x	product	total earn
0	Brazil	Ammonia, anhydrous	24144572.05	399156.28	23745415.77	2571549.00	0.924488	rice	2.781594e+06
1	Brazil	Ammonium nitrate (AN)	21546416.92	214625.65	21331791.27	13821890.37	4.969055	rice	2.781594e+06
2	Brazil	Ammonium sulphate	30988444.53	53840.25	30934604.28	22805149.59	8.198592	rice	2.781594e+06
3	Brazil	Calcium ammonium nitrate (CAN) and other mixtu...	2020987.02	10844.03	2010142.99	1164236.84	0.418560	rice	2.781594e+06
4	Brazil	Diammonium phosphate (DAP)	6948831.81	148854.13	6799977.68	4846157.53	1.742223	rice	2.781594e+06
...	...	...	...	...	...	...	...	...	...
455	Mexico	Sodium nitrate	32779.02	1030.91	31748.11	0.00	0.000000	WHEAT	5.106931e+05
456	Mexico	Superphosphates above 35%	622825.23	2467787.77	-1844962.54	811919.38	1.589838	WHEAT	5.106931e+05
457	Mexico	Superphosphates, other	0.00	0.00	0.00	21682383.00	42.456774	WHEAT	5.106931e+05
458	Mexico	Urea	21917270.86	303460.75	21613810.11	8931655.72	17.489281	WHEAT	5.106931e+05

شکل أ فراخوانی دیتاست. سایز دیتاست به اندازه ٤٦٠ در ۹ می‌باشد.

در گام بعد هر کرومزوم بصورت زیر خواهد بود که شامل ۱۸ ژن می‌باشد. کود در آلل (allel) شماره ۰، کودی با بیشترین اهمیت از نظر دسترس پذیری و بهروری خواهد بود. اگر کشوری شامل کودی نباشد در محاسبه fitness کود محاسبه نخواهد شد. کرومزوم زیر کرومزومی فرضی است که در آن کود شماره یک دارای بیشترین اهمیت و کود شماره ۱۸ دارای کمترین اهمیت است. در اصل ما ۲۳ کود داریم که فرض شده ۵ تا از این کودها باهم مقایرت دارند و در اثر ترکیب اثرات مخربی خواهند داشت پس در هر کرومزوم حداکثر یکی از این کودها می‌تواند موجود باشد.

```
conflict = np.array([1,23,14,10,19])
```

کود شماره یک	کود شماره دو	کود شماره سه	کود شماره چهار	...	کود شماره هفده	کود شماره هجده

۲. تعریف مسئله با تابع پنالتی

در این بخش ما به تعریف مسئله خواهیم پرداخت. در pymoo نیاز است که تعداد متغیرها، اهداف و حد بالا و پایین متغیرها تعریف شود. در این قسمت با استفاده از تابع super این کار انجام شده که برای تمام الگوریتم‌های ثابت است در این مسئله خاص. مسئله ما دارای ۲۳ متغیر(خط ٤)، دو هدف (خط ٥)، بدون قید (خط ٦)، و کران پایین ۰ و کران بالای ۲۳ برای هر متغیر (خط ۷و ۸) است. سپس قسمت evaluate خواهد بود که شامل مینیمم کردن دو هدف f_1 و f_2 است (مسئله اصلی ماکزیمم کردن این دو که با ضرب یک منفی با مینیمم کردن در واقع ماکزیمم را بدست می‌آوریم.). برای هر کشور چک می‌شود و اهمیت هر کود با توجه به مکانی که قرار دارد با مقادیر قبل جمع می‌شود. باید خاطر نشان کرد که در این کتابخانه اول تمامی جمعیت به این تابع داده می‌شود پس نیاز است آرایه‌ای به اندازه تعداد جمعیت برای f_1 و f_2 درنظر گرفته بشود (خط۱۱ و ۱۲). در آخر f_1 (دسترس پذیری) وf_2 (بهروری) در آرایه خط ۲۳ کنار هم به ازای هر عضو جمعیت قرار می‌گیرند.

اما اینجا ما دو قید داریم که برای handle کردن این قیود از پنالتی باینری استفاده شده است. در قید اول g_1 که در واقع همان کنترل نیامدن کودهای مقایر در کرومزوم که خطوط ۱۹، ۲۰، ۲۱ و ۲۲ همچنین ۳۱ مربوط هستند به این قید. طرز کار به اینصورت است که نگاه میکند اگر ژنی در آرایه conflict آمده باشد آنگاه یک واحد به flag اضافه میکند و بعد به ژن بعد رفته اگر دوباره ژنی در conflict باشد در این کرومزوم آنگاه در خط ۳۱ جریمه میکند این عضو را و flag برای هر عضو reset می‌گردد.

در handle قید دوم g_2 که اگر کودی در بهروری محصول اثر نداشته باشد نمی‌تواند در کرومزوم موجود باشد پس باید جریمه بشود. در خط ۲۹ و ۳۰ می‌توان آن را دید.

```
1  class MyProblem(Problem):
2
3      def __init__(self):
4          super().__init__(n_var=18,
5                           n_obj=2,
6                           n_constr=0,
7                           xl=np.array([0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]),
8                           xu=np.array([23,23,23,23,23,23,23,23,23,23,23,23,23,23,23,23,23,23]))
9
10     def _evaluate(self, X, out, *args, **kwargs):
11         f1 = np.zeros(X.shape[0])
12         f2 = np.zeros(X.shape[0])
13         g1 = np.ones(X.shape[0])
14         g2 = np.ones(X.shape[0])
15         flag = 0
16         for p in range (X.shape[0]):
17             flag = 0
18             for i in range (18):
19                 temp = np.where(conflict == X[p,i])
20                 temp = temp[0]
21                 if temp.size == 0:
22                     flag += 1
23
24                 for j in range (459):
25                     if samples[j,1] == X[p,i] and samples[j,7] == 'rice':
26                         f1[p] += -(samples[j,4] * (18 - i))
27                         f2[p] += -(samples[j,6] * (18 - i))
28                     # print('hi')
29                     if samples[j,6] == 0.0:
30                         g2[p] = 0
31             if flag > 1:
32                 g1[p] = 0
33
34
35         out["F"] = np.column_stack([f1, f2])
36         out["G"] = np.column_stack([g1, g2])
37
38 problem = MyProblem()
```

شکل ب. تعریف مسئله با تابع پنالتی

۲-۱. *الگوریتم‌ها*

این قسمت بسیار ساده و صریح است. تنها نیاز است الگوریتم فراخوانی و import شود (شکل ت). با الگوریتم NSGA۲ آغاز میکنیم نیاز است از

Myproblem در بخش قبل یک شی ساخته شود (خط ۱). در خط دوم مقادیری به الگوریتم داده به ترتیب، اندازه جمعیت (برابر ۱۰۰)، نوع representation (در اینجاpermutation که اول بصورت رندوم)، سپس crossover (در اینجا تغییر order)، سپس جهش (در این جا برعکس کردن).

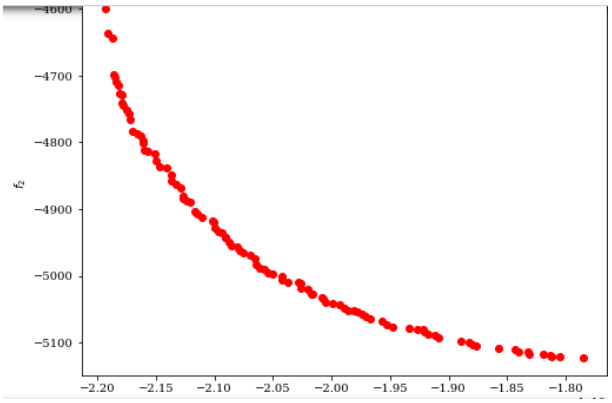
جدول أ. پارامترهای مورد استفاده

Representation	Permutation
crossover	Order
mutation	Inverse
Population size	۱۰۰
Initial population	Random
Termination	۲۰۰ generation

```
from pymoo.algorithms.nsga2 import NSGA2
algorithm = NSGA2(pop_size=100, sampling=get_sampling("perm_random"),
                  crossover=get_crossover("perm_ox"),
                  mutation=get_mutation("perm_inv"),
                  eliminate_duplicates=True)
res_NSGA2 = minimize (problem, algorithm, Termination = ("n_gen", 200),
                      seed = 1, verbose = False, save_history=True)
#print(res_NSGA2.F)
plot = Scatter()
plot.add(problem.pareto_front(), plot_type="line", color="black", alpha=0.7)
plot.add(res_NSGA2.F, color="red")
plot.show()
```

شکل ت. فراخوانی الگوریتم NSGA۲

نتایج در خط ٦ ذخیره می‌شود. خطوط ۷، ۸، ۹ و ۱۰ نمودار pareto front را برای ما رسم خواهند کرد (شکل ث).



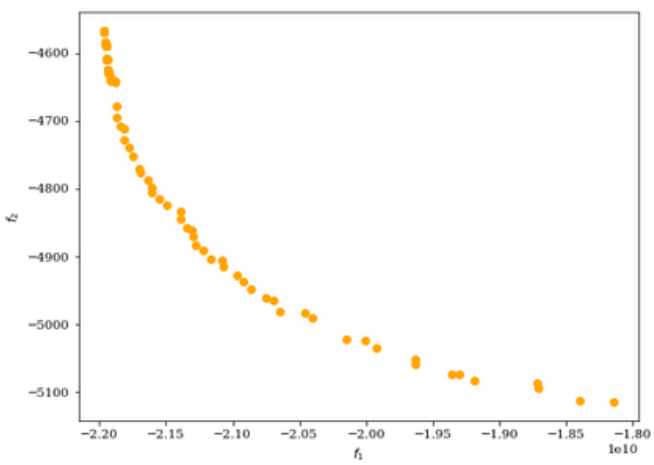
شکل ث. نمودار pareto front برای NSGA۲ (محور عمودی بهروری و محور افقی دسترس پذیری)

حال به بررسی الگوریتم RSNGA۲ می‌پردازیم. پارامترها شبیه به جدول أ، استفاده خواهد شد تنها اینجا ref points با مقادیر قابل مشاهده در خط ۳ و همچنین مقدار اپسیلون به پیشنهاد خود مقاله ۰.۰۱ انتخاب شده. نتیجه pareto front در شکل ح مشاهده می‌کنید.

```
from pymoo.algorithms.rnsga2 import RNSGA2

ref_energy = get_reference_directions("energy", 2, 100, seed=1)
algorithm_energy = RNSGA2(pop_size=100, sampling=get_sampling("perm_random"),
                  crossover=get_crossover("perm_ox"),
                  mutation=get_mutation("perm_inv"),
                  eliminate_duplicates=True,
                  ref_points = ref_energy, epsilon=0.01,
                  normalization='front')
res_RNSGA2_energy = minimize (problem, algorithm_energy, Termination = ("n_gen", 200),
                              seed = 1, verbose = False, save_history=True)
plot = Scatter()
plot.add(problem.pareto_front(), plot_type="line", color="black", alpha=0.7)
plot.add(res_RNSGA2_energy.F, color="Orange")
plot.show()
```

شکل ج. الگوریتم RNSGA۲



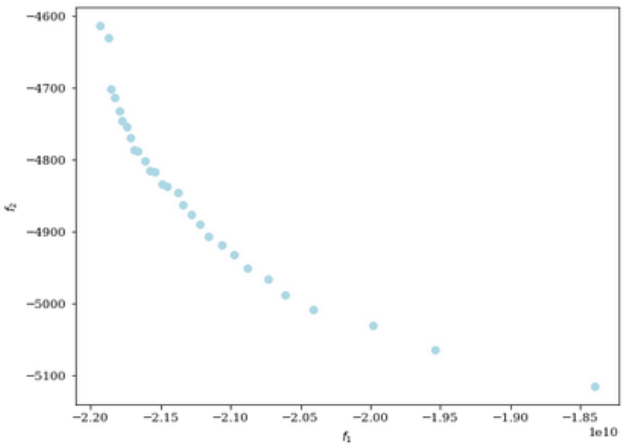
شکل ح. نمودار pareto front برای RSNGA۲ (محور عمودی بهروری و محور افقی دسترس پذیری)

الگوریتم بعدی NSGA۳ است. اینجا باید به ۱۰۰ reference direction داشته باشیم (خط ۲) و بقیه پارامترها مشابه جدول أ، است.

```
from pymoo.algorithms.nsga3 import NSGA3
ref_energy = get_reference_directions("energy", 2, 100, seed=1)
algorithm_energy = NSGA3(pop_size=100, sampling=get_sampling("perm_random"),
                  crossover=get_crossover("perm_ox"),
                  mutation=get_mutation("perm_inv"),
                  eliminate_duplicates=True,
                  ref_dirs=ref_energy,
                  normalization='front')
res_NSGA3_energy = minimize (problem, algorithm_energy, Termination = ("n_gen", 200),
                             seed = 1, verbose = False, save_history=True)
plot = Scatter()
plot.add(problem.pareto_front(), plot_type="line", color="black", alpha=0.7)
plot.add(res_NSGA3_energy.F, color="lightblue")
plot.show()
```

شکل خ. فراخوانی الگوریتم NSGA۳

در شکل زیر pareto front این الگوریتم دیده می‌شود.



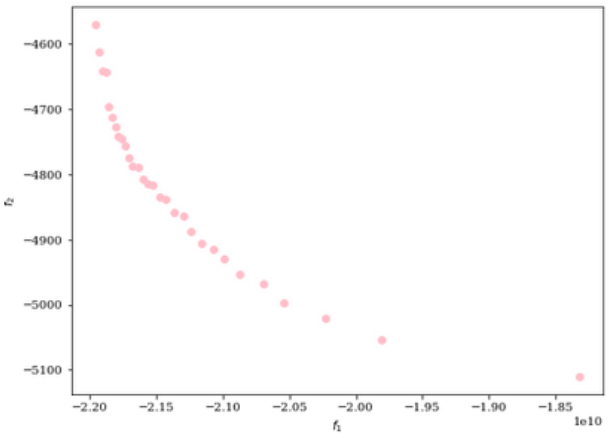
شکل د. نمودار pareto front برای NSGA۳ (محور عمودی بهروری و محور افقی دسترس پذیری)

در الگوریتم UNSGA۳ می‌پردازیم. پارامترها شبیه به جدول أ، استفاده شده‌اند. و reference direction نیز در خط ۲ تعریف شده است.

```
from pymoo.algorithms.unsga3 import UNSGA3
ref_energy = get_reference_directions("energy", 2, 100, seed=1)
algorithm = UNSGA3(pop_size=100, sampling=get_sampling("perm_random"),
                  crossover=get_crossover("perm_ox"),
                  mutation=get_mutation("perm_inv"),
                  eliminate_duplicates=True,
                  ref_dirs=ref_energy,
                  normalization='front')
res_UNSGA3_energy = minimize (problem, algorithm, Termination = ("n_gen", 200),
                              seed = 1, verbose = False, save_history=True)
plot = Scatter()
plot.add(problem.pareto_front(), plot_type="line", color="black", alpha=0.7)
plot.add(res_UNSGA3_energy.F, color="pink")
plot.show()
```

شکل ذ. الگوریتم UNSGA۳

و pareto front در شکل زیر آمده است.



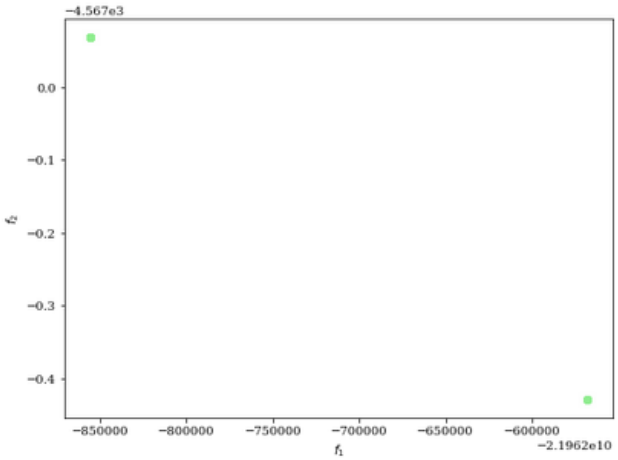
شکل ر. نمودار pareto front برای RSNGA۲ (محور عمودی f۲ و محور افقی f۱)

در الگوریتم MOEAD نیز روال به همان منوال است منتها در این الگوریتم نتیجه مطلوبی به هیچ وجه مشاهده نشد.

```
from pymoo.algorithms.moead import MOEAD
algorithm_energy = MOEAD(get_reference_directions("energy", 3,  100),
                  n_neighbors=20,
                  decomposition="pbi",
                  prob_neighbor_mating=0.7,pop_size = 100,seed=1,sampling=get_sampling("perm_random"),
                  crossover=get_crossover("perm_ox"),mutation=get_mutation("perm_inv"),eliminate_duplicates=Tru
res_MOEAD_energy = minimize(problem, algorithm_energy, termination=('n_gen', 200)
                            , verbose = False, save_history=True)
plt = Scatter()
plt.add(problem.pareto_front(), plt_type="line", color="black", alpha=0.7)
plt.add(res_MOEAD_energy.F, color="lightgreen")
#plt.add(ref_points)
plt.show()
```

شکل ز. الگوریتم MOEAD

و pareto front این الگوریتم را در شکل زیر آمده است و تعداد اعضا بسیار نامناسب است. به این امید که وضع مجموعه پاسخ‌ها بهتر بشود از reference point سه بعدی انتخاب کردیم اما در پاسخ‌ها بهبودی مشاهده نشد.



شکل س. نمودار pareto front برای MOEAD (محور عمودی f۲ و محور افقی f۱)

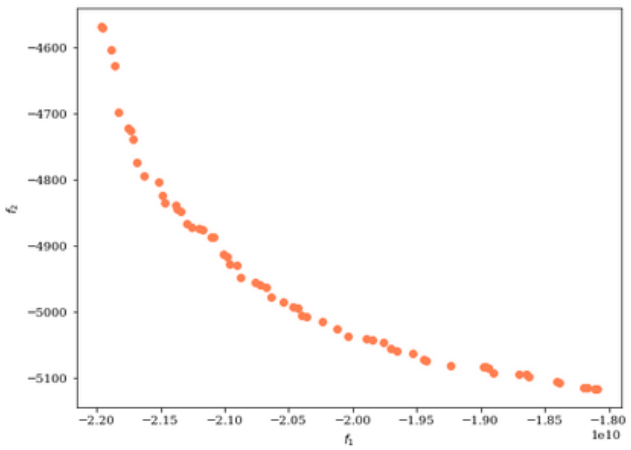
الگوریتم بعدی که مورد استفاده قرار گرفته است الگوریتم CTAEA است. در این الگوریتم تمامی پارامترها به مانند قبل انتخاب شده است. در pareto front این الگوریتم مشاهده می‌گردد که تعداد پاسخ‌ها از MOEAD بهتر است.

```
from pymoo.algorithms.ctaea import CTAEA

ref_dirs = get_reference_directions("energy", 2, 100)

algorithm = CTAEA(sampling=get_sampling("perm_random"),
                  crossover=get_crossover("perm_ox"),
                  mutation=get_mutation("perm_inv"),
                  eliminate_duplicates=True,
                  ref_dirs=ref_dirs)
res_CTAEA = minimize(problem, algorithm, termination=('n_gen', 200), verbose = False, save_history=True)
plt = Scatter()
plt.add(problem.pareto_front(), plt_type="line", color="black", alpha=0.7)
plt.add(res_CTAEA.F, color="coral")
#plt.add(ref_points)
plt.show()
```

شکل ش. الگوریتم CTAEA



شکل ص. نمودار pareto front برای CTAEA (محور عمودی f۲ و محور افقی f۱)

الگوریتم آخر RNSGA۳ می‌باشد. که باز هم مانند قبل بازنمایی و جهش و... طبق جدول أ است. و reference point هایی که در خط ۳ می‌بینید.

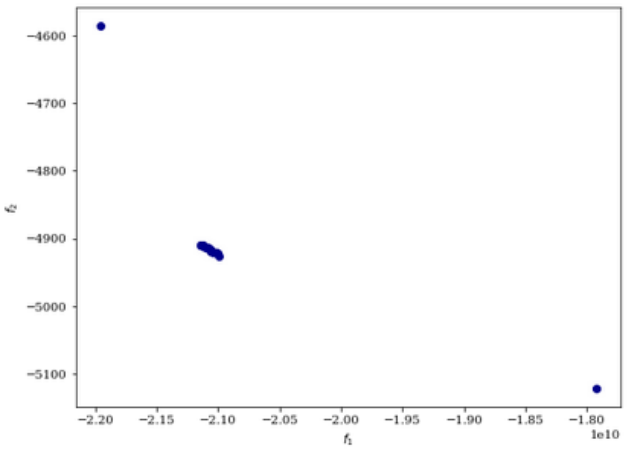
```
from pymoo.algorithms.rnsga3 import RNSGA3

ref_energy = get_reference_directions("energy", 2, 12)
algorithm_energy = RNSGA3(ref_points = ref_energy, pop_per_ref_point = 90,
                  sampling=get_sampling("perm_random"),
                  crossover=get_crossover("perm_ox"),
                  mutation=get_mutation("perm_inv"),
                  eliminate_duplicates=True)
res_energy =  minimize (problem, algorithm_energy, Termination = ("n_gen", 200),
                        seed = 1, verbose = False, save_history=True)
plt = Scatter()
plt.add(problem.pareto_front(), plt_type="line", color="black", alpha=0.7)

plt.add(res_energy.F, color="darkblue")
#plt.add(ref_points)
plt.show()
```

شکل ض. الگوریتم RNSGA۳

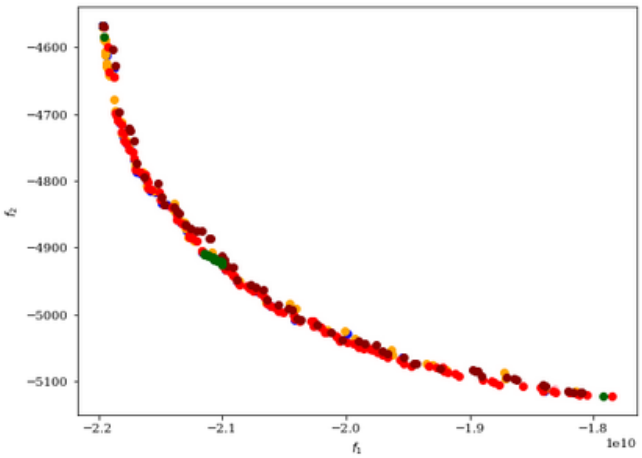
در شکل زیر pareto front این الگوریتم مشاهده می‌شود. که نتیجه زیاد مطلوب نیست اما از MOEAD بهتر است.



شکل ط. . نمودار pareto front برای RNSGA۳ (محور عمودی f۲ و محور افقی f۱)

۲-۲. نتیجه‌گیری

در این قسمت با مقایسه pareto front الگوریتم‌ها متوجه می‌توان شد که تعداد زیادی از الگوریتم‌ها pareto front مشابه داشتند اما بطور کل MOEAD عملکرد بدتری داشت و NSGA۲ که قرمز است در شکل زیر عملکرد خوبی داشت.



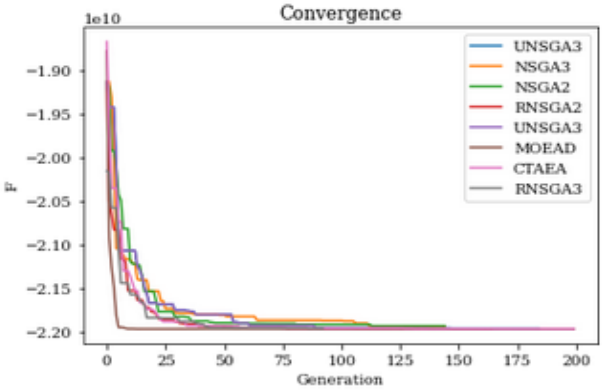
شکل ظ. مقایسه pareto front الگوریتم‌ها. (محور عمودی f۲ و محور افقی f۱). نقاط قرمز مربوط به NSGA۲، صورتی UNSGA۳، آبی NSGA۳، نارنجی RNSGA۲، قرمز تیره یا همان قهوه‌ای CTAEA، آبی تیره MOEAD و در آخر سبز تیره RNSGA۳.

```
plot = Scatter()
plot.add(problem.pareto_front(), plot_type="line", color="black", alpha=0.7)
plot.add(res_UNSGA3_energy.F, color="pink", label="UNSGA3")
plot.add(res_NSGA3_energy.F, color="blue", label="NSGA3")
plot.add(res_RNSGA2_energy.F, color="Orange", label="RNSGA2")
plot.add(res_NSGA2.F, color="red", label="NSGA2")
plot.add(res_MOEAD_energy.F, color="darkblue", label="MOEAD")
plot.add(res_CTAEA.F, color="darkred", label="CTAEA")
plot.add(res_energy.F, color="darkgreen", label="RNSGA3")

plot.show()
```

شکل ع. کد مربوط به تمامی pareto frontها

همچنین از لحاظ همگرایی الگوریتم‌ها بصورت شکل زیر می‌باشند. که همگرایی MOEAD از همه بهتر است. اما چون pareto front مطلوبی ارائه نمی‌کند. RNSGA۳ از همه بهتر است در همگرایی خط مشکی.



شکل غ مقایسه همگرایی الگوریتم‌ها. محور افقی نمایانگر تعداد نسل‌ها و نمودار عمودی نمایانگر هدف است.

۳. تعریف مسئله با تعمیر

این بخش نیز تمامی مسائل قبل از قیدش مانند بخش ۲ است اما در قید بجای penalty ما کرمزوم را repair کردیم. به این صورت که در خطوط ۲۳ تا ۳۰ تامین شده به اینصورت که اگر کرومزومی شامل دو عضو از conflict باشد بصورت random یک عدد کود انتخاب شده و چک می‌شود که نه در کرومزوم و نه در conflict موجود باشد. در قید دوم که مربوط به بهروری بود باید در این قید دوباره کودی که در بهروری برنج یا هر محصول دیگر تاثیر ندارد حذف گردد و جای خود را به موثر بدهد.

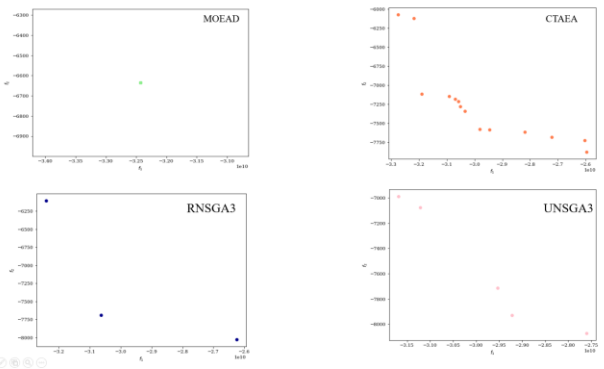
```
def _evaluate(self, X, out, *args, **kwargs):
    f1 = np.zeros(X.shape[0])
    f2 = np.zeros(X.shape[0])
    g1 = np.ones(X.shape[0])
    g2 = np.ones(X.shape[0])
    flag = 0
    for p in range (X.shape[0]):
        flag = 0
        for i in range (17):
            temp = np.where(conflict == X[p,i])
            temp = temp[0]
            if temp.size != 0:
                while (True):
                    num = random.randrange(0,22)
                    temp = np.where(conflict == num)
                    temp = temp[0]
                    temp2 = np.where(X[p,:] == num)
                    temp2 = temp2[0]
                    if temp.size == 0 and temp2.size == 0:
                        X[p,i] = num
                        break
            for j in range (459):
                if samples[j,1] == X[p,i] and samples[j,7] == 'rice':
                    if samples[j,6] == 0.0:
                        while (True):
                            num = random.randrange(0,22)
                            temp = np.where(conflict == num)
                            temp = temp[0]
                            temp2 = np.where(X[p,:] == num)
                            temp2 = temp2[0]
                            if temp.size == 0 and temp2.size == 0:
                                X[p,i] = num
                                break

                    else:
                        f1[p] += -(samples[j,4] * (18 - i))
                        f2[p] += -(samples[j,6] * (18 - i))
```

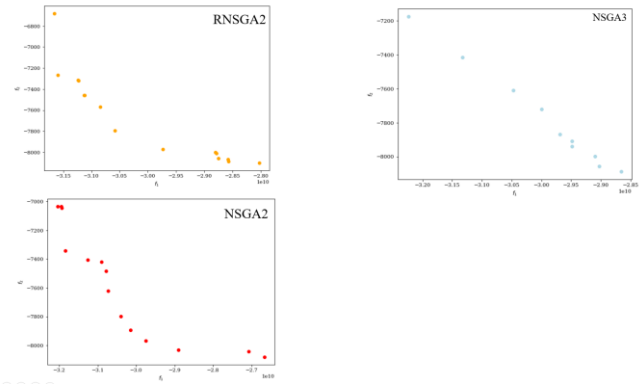
شکل ف. عریف مسئله با repair

۳-۱. الگوریتم‌ها

تمامی کدها با الگوریتم‌های بخش قبل دست نخورده باقی مانده است این قسمت تنها به مشاهده و مقایسه نتایج خواهیم پرداخت.



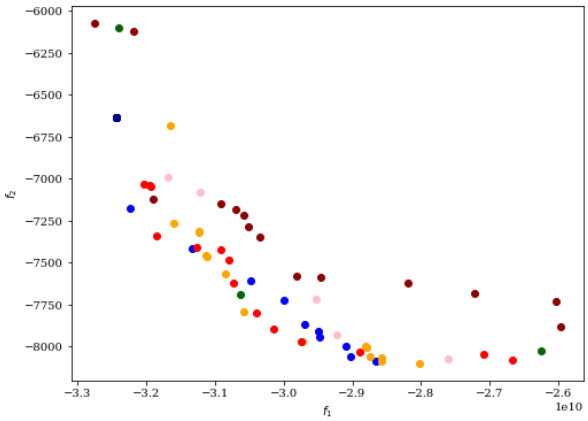
شکل ق. نتایج حاصل از اجرای مسئله با قید با استفاده از repair روی الگوریتم‌های RNSGA۳، CTAEA، UNSGA۳ و MOEAD



شکل ک. ن. نتایج حاصل از اجرای مسئله با قید با استفاده از repair روی الگورتیم‌های NSGA۲، RNSGA۳ و NSGA۳ می‌باشد.

۳-۲. نتیجه‌گیری

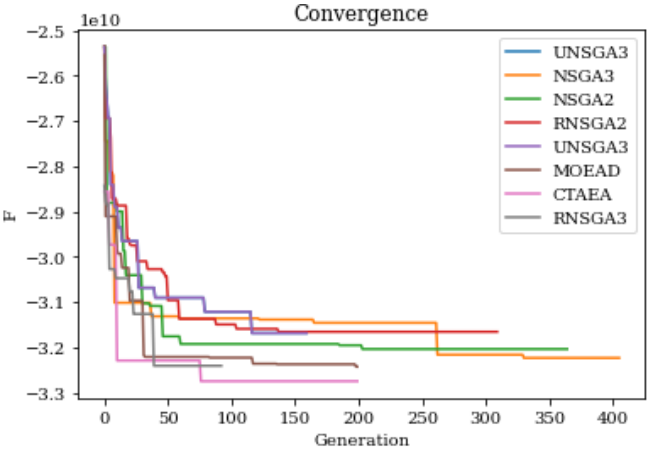
در این قسمت میتوان با استفاده از نتیجه گیری حل مسئله قبل متوجه این شد که این مسئله با استفاده از repair مناسب جواب نمی‌دهد. کد مانند شکل ع است.



شکل ل. مقایسه pareto front الگوریتم‌ها. (محور عمودی f۲ و محور افقی f۱). نقاط قرمز مربوط به NSGA۲، صورتی UNSGA۳، آبی NSGA۳، نارنجی RNSGA۲، قرمز تیره یا همان قهوه‌ای CTAEA، آبی تیره MOEAD و در آخر سبز تیره RNSGA۳.

با مقایسه شکل لشکل ظ می‌توان نتیجه گرفت این مسئله با پنالتی جواب‌های بهتر و بیشتری می‌دهد.

درباره همگرایی نیز می‌توان گفت CTAEA از همه عملکرد بهتری داشت و زودتر همگرا شده است (روی مسئله حل کردن قید با repair).



شکل م. مقایسه همگرایی الگوریتم‌ها. محور افقی نمایانگر تعداد نسل‌ها و نمودار عمودی نمایانگر هدف است