دانشکده مهندسی

گروه مهندسی کامپیوتر

گرایش هوش مصنوعی

بهینه سازی چند هدفه بر اساس تنظیم تطبیقی وزن های و همسایگی ها

استاد:

دکتر مجتبی روحانی

دانشجو:

علی گلی

اسفند ماه

سال ۱۴۰۲

۱ معرفی

در نسخه های اولیه MOEAD ابتدا تعدادی بردار های وزن ساخته می‌شود. هر زیر مسئله همسایگی های یکسان و منابع محاسباتی برابری هم دارد. این نوع MOEAD کمبود های خاصی دارد که زیر مسئله های متفاوت مشکلات متفاوت خاص خود را در رسیدن به سطح pareto دارند.اگر به هر زیر مسئله منابع یکسانی تخصیص داده شود، مسائل دشوار دچار کمبود منابع و مسائل ساده دچار منابع بیش از اندازه می‌شوند.

در این نوع بهینه سازی مشکلات دیگری من جمله همگرایی و گوناگونی نیز مطرح است. برای حل این مشکلات روش بهینه سازی چند هدفه بر اساس تنظیم تطبیقی وزن های و همسایگی ها پیشنهاد داده شده است. (MOEAD-AAWN: MOEA/D with adaptively adjusting weight vectors and neighborhood). در ابتدا یک روش جدید در ساخت وزن های پیشنهاد شده است که تنوع همسایگی را افزایش دهد. به دنبال آن یک روش جهت تطابق همسایگی ها پیشنهاد شده است که کمک می‌کند تا با استفاده از جستجوی کلی به جستجوی محلی بپردازیم. همچنین همسایگی ها با توجه به موقعیت بردارهای وزن خود قرار می‌گیرند. مقاله معرفی شده در ۳ بخش دارای نوآوری می‌باشد:

- جهت هر زیرمسئله آنالیز می‌گردد و با استفاده از روشی به نام Spa میزان تراکم جمعیت روی سطح پرتو اندازه گیری می‌شود.
- با استفاده از Spa یک روش تطبیق دادن بردار های وزن پیشنهاد شده است تا گوناگونی جمعیت نهایی برآورده شود.
- یک روش برای تنظیم تطبیقی همسایگان برای تغییر جستجوی سراسری به جستجوی محلی. با این روش و با این فرض که جواب نهایی گوناگون خواهد بود مقدار زمان اجرا نیز کاهش می‌یابد.

۲ مقدمات

۲.۱ تعریف مسئله

در تعریف مسئله در نظر می‌گیریم یک MOP با n متغیر و m هدف وجود دارد.

$$\min \ F(x) = (f_1(x), f_2(x), \ldots, f_m(x)), \ x \in \Omega \qquad (1)$$

که $x = (x_1, \ldots, x_n) \in \Omega = \prod_{i=1}^{n} [a_i, b_i] \subset R^n$, Ω فضای تصمیم ما می‌باشد و $F(x) = (f_1, \ldots, f_m) : \Omega \rightarrow R^m$ یک تابع هدف m بعدی می‌باشد.

از آنجایی که توابع مختلف در مسائل MOP با همدیگر ناسازگاری هایی دارند، رسیدن به به یک جوابی که برای همه توابع هدف بهینه باشد غیرممکن است. برای مقایسه کارایی جواب های متفاوت مفاهیم زیر معرفی می‌شوند.

تعریف اول (غلبه پرتو): اگر x,y هردو عضوی از راه حل های مسئله بهینه سازی باشند به شرطی که:

1) $\forall i \in \{1, 2, \ldots, m\}$, s.t. $f_i(x) \leq f_i(y)$ and
2) $\exists j \in \{1, 2, \ldots, m\}$, s.t. $f_j(x) < f_j(y)$.

می‌گوییم که x غلبه کرده است بر y و به شکل روبرو آن را می‌نویسیم: $x \prec y$

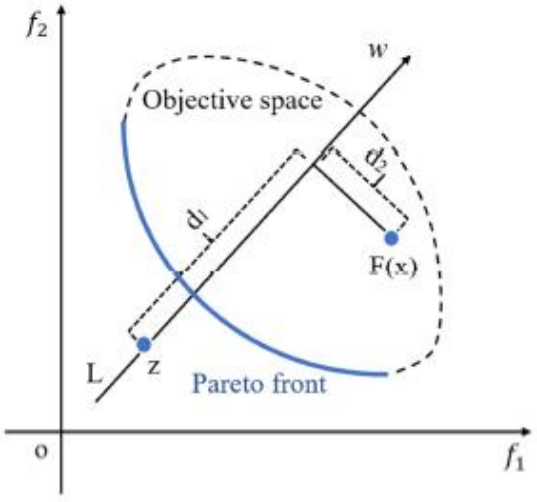
تعریف دوم (جواب بهینه پرتو): x عضو سطح پرتو است اگر و فقط اگر به ازای هیچ y رابطه $x \prec y$ صحیح باشد.

تعریف سوم (مجموعه پرتو و PF): مجموعه پرتو، مجموعه ای شامل تمامی جواب های بهینه پرتو است در فضای تصمیم و به عمل نگاشت مجموعه پرتو در فضای اهداف PF گفته می‌شود.

۲.۲ روش های Decomposing توابع هدف در MOEA/D

MOEAD در ابتدا یک مسئله MOP را به تعدادی زیرمسئله تقسیم می‌کند تا آنها بتوانند جواب های بهینه سطح پرتو را دریافت کنند. روش های گفته شده تا کنون شامل روش های Weighted Sum و یا Tchebycheff و penalty-based boundry intersection (PBI) می‌باشد. در این مقاله از روش PBI استفاده می‌شود. روش PBI را می‌توان از شکل و فرمول های زیر بدست آورد:

$$\begin{aligned} \min \quad & g^{\mathrm{PBI}}(x|w,z) = d_1 + \theta d_2 \\ \% \quad & d_1 = \frac{||(z - F(x))^T w||}{||w||} \\ & d_2 = \|F(x) - (z + d_1 w)\| \end{aligned}$$



که در آن w و z به ترتیب بردارهای وزن و نقطه رفرنس هستند. فاصله d_1, d_2 به ترتیب از چپ به راست نشان دهنده تصویر فاصله F(x) تا z روی بردار w و فاصله w تا z می‌باشد.

۳ روش های پیشنهادی

۳.۱ روش تنظیم تطبیقی بردار های وزن

با توجه به مشکلات بیان شده؛ اول اینکه بردار های وزن نمی‌توانند تضمین کنند که جواب های بهینه بر روی PF به صورت یکنواخت در نواحی تیز و دنباله یکسان باشد و دوم اینکه برای MOP هایی که PF آنها ناپیوسته است، زیرمسائل مختلف ممکن است جواب های بهینه یکسان داشته باشند که باعث تخریب اصل گوناگونی می‌شود. برای اینکه جواب های گوناگون داشته باشیم بردار های وزن با توجه به موقعیت خود باید تنظیم شوند. برای تنظیم بردار های وزن می‌توانیم از تابع sparsity استفاده کنیم.

۱. تابع sparsity: این تابع با توجه به مقادیر توابع هدف هر زیر مسئله تعریف می‌شود. برای k امین زیر مسئله جواب x^k پیدا شده است که مقدار آن در توابع هدف برابر $F(x^k)$ است. این جواب بهینه با جواب های بهینه سایر زیر مسائلی مقایسه می‌شود که در همسایگی زیر مسئله ما قرار دارند. فرمول محاسبه

$$Spa(k) = \min_{i=1}^{T} \sqrt{(F(x^k) - F(x^i))(F(x^k) - F(x^i))^T}$$
$$= \min_{i=1}^{T} \sqrt{\sum_{j=1}^{m} (f_j(x^k) - f_j(x^i))^2}$$

که K = 1,2, ... , N که همگی همسایگی هستند. هرچقدر که spa یک زیر مسئله بزرگتر باشد نشان می‌دهد که محاسبات تکاملی این زیرمسئله از بقیه متفاوت تر بوده است. spa فاصله در فضای توابع هدف را انجام می‌دهد. برای هرس کردن وزن هایی که باعث شده است تا در فضای توابع جواب های بهینه بسیار به یکدیگر نزدیک شوند یک روش پیشنهاد شده است.

۲. هرس کردن بردار های وزن: برای زیر مسئله K ام، اگر Spa آن مینیمم باشد، وزن مربوط به آن حذف خواهد شد. در مقابل دو زیرمسئله ای که ماکسیمم مقادیر Spa را داشته باشند برای ساخت وزن جدید استفاده می‌شوند:

$$w_{\text{new}} = (w_k + w_l)/2$$

Algorithm 1 Method of Adaptively Adjusting Weight Vectors

Input: N: Population size
gen_max: Maximum number of iterations
m: The number of objective functions
$t = 0$
Output: New weight vectors $W = w^1, w^2, ..., w^N$

while $t < gen\_max$ **do**
  $t = t + 1$
  $k = 0$
  while $k < N$ **do**
    $k = k + 1$
    Find the Pareto optimal solution x^k for k^{th} subproblem.
    Compute the Spa of subproblems:

$$Spa(k) = \min_{i=1}^{T} \sqrt{\sum_{j=1}^{m} (f_j(x^k) - f_j(x^i))^2}$$

  end while
  Find the minimum Spa value of subproblem and delete its corresponding weight vector.
  Find the maximum Spa value of subproblem and add a new weight vector:

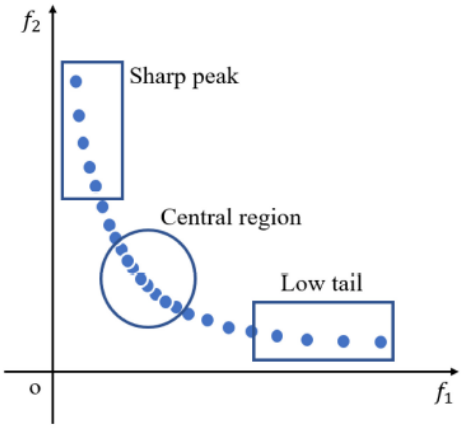
$$w_{new} = (w_k + w_l)/2$$

  if $||max(Spa) - min(Spa)|| < \epsilon$ **then**
    return
  end if
end while

۳.۲ روش تنظیم تطبیقی همسایگی

در MOP زیر مسئله های متفاوت پیچیدگی های مختلفی دارند. به طور مثال منابع محاسباتی هدر خواهند رفت اگر همسایگی های یکسانی در MOEAD استفاده شوند.

به طور مثال در شکل بالا که توسط یکسری از وزن های یکنواخت، جواب های بهینه تولید شده اند، این جواب ها در نقاط تیز و دنباله بسیار گسترده و پراکنده هستند ولی در نقاط مرکزی بسیار فشرده هستند.



هرچقدر جواب ها به نواحی مرزی نزدیک‌تر می‌شوند، پراکنده تر می‌شوند. در MOEA/D ساده تمامی زیرمسائل در هر نسل همسایگان یکسانی دارند. در حالی که زیر مسائل در نسل های مختلف پیچیدگی های محاسباتی مختلفی نیز دارند. همین موضوع متفاوت بودن در میزان زمان محاسبات باعث کاهش چگالی در نقاط تیز و دنباله ها شده است. باید روشی را استفاده کنیم که نقاط بهینه در قسمت های تیز و دنباله های PF نیز حداکثر گردد و از بهینه های محلی به سمت بهینه های سراسری حرکت کنیم.

همسایگی ها در کیفیت الگوریتم نقش بسیار مهمی را دارند. همسایگی های بزرگ باعث افزایش تنوع می‌شوند و همسایگی های کوچک باعث کاهش بار محاسباتی می‌شوند.

۱. اولین تنظیم همسایگی ها: در ابتدا محاسبات تکاملی تعداد همسایگی ها بزرگ می‌ماند تا همگرایی الگوریتم به صورت سراسری شتاب پیدا کند. همانطور که الگوریتم تکاملی رشد پیدا می‌کند، تعداد همسایگی جهت افزایش همگرایی در ناحیه محلی کاهش پیدا می‌کند. اولین تنظیم همسایگی از طریق فرمول زیر صورت می‌گیرد:

$$T_i' = \left(1 - \alpha \frac{\text{gen}}{\text{gen\_ max}}\right) T_i,\ i = 1, 2, \ldots, N$$

که در آن gen مربوط به عدد نسل فعلی و gen_max حداکثر نسل می‌باشد. T_i نیز شان دهنده همسایگی زیر مسئله iام می‌باشد.

۲. دومین تنظیم همسایگی ها: همسایگی ها در فضاهای اسپارس کاهش پیدا می‌کنند و در فضاهای متراکم افزایش پیدا می‌کنند. برای افزایش کیفیت تکاملی با توجه به موقعیت راه حل ها روی PF یک روش دومی ارائه می‌شود که بر اساس تابع Spa همسایگی هارا تنظیم می‌نماید.

$$T_i'' = \left(1 - \beta\left(\frac{\text{Spa}(k)}{\max \text{Spa}(k)}\right)\right) T_i',\ i = 1, 2, \ldots, N$$

که در آن Spa(k) نشان دهنده تابع sparsity در زیرمسئله kام می‌باشد. maxSpa(k) نشان دهنده حداکثر sparsity برای همه زیرمسئله هاست.

Algorithm 2 Method of Adaptively Adjusting Neighborhoods

Input: N: Population size
T: Neighborhoods of subproblems
m: The number of objective functions
$gen\_$ max: Maximum number of iterations
$t = 0$

Output: T new neighborhoods

initialize $B(i) = \{i_1, i_2, ..., i_T\}$.
while $t < gen\_$ max **do**
  $t = t + 1$
  $i = 1$
  while $i \leq N$ **do**
    Implement strategy of first adjustment:

$$T_i' = (1 - \alpha \frac{gen}{gen\_ \max}) T_i, i = 1, 2, ..., N$$

    Calculate the Spa of subproblems.
    Implement strategy of second adjustment:

$$T_i'' = (1 - \beta(\frac{Spa(k)}{\max Spa(k)})) T_i', i = 1, ..., N$$

    $i = i + 1$
  end while
end while

۴ فلوچارت الگوریتم

قبل از ایجاد عملیات اولیه، تعداد توابع هدف برابر m، تعداد زیرمسئله ها برابر n، تعداد همسایگی های زیر مسئله ها برابر T و حداکثر نسل برابر max_gen می‌باشد.

بردارهای وزن به صورت رندوم توسط الگوریتم تولید می‌شوند و سپس بردار های هر زیر مسئله $w^1, w^2, \dots, w^N$ توسط الگوریتم اول محاسبه می‌شود. همچنین $z = (z_1, z_2, \dots, z_m)^T$ توسط فرمول روبرو مقدار دهی اولیه می‌گردد:

$$z_i = \min f_i(x_j), j = 1,2, \dots, N.$$

در فاز بروزرسانی، همسایگی زیرمسئله iام توسط الگوریتم دوم به صورت تطبیقی بروزرسانی می‌گردد. روند بروزرسانی به شکل زیر می‌باشد:

Algorithm 3 MOEA/D-AAWN

Input: N: Population size
T: Neighborhoods of subproblems
$B(i) = \{i_1, i_2, ..., i_T\}$ Neighbor of individual i
$gen\_$ max: Maximum number of iterations
$t = 0$
Output: EP external populations

Initialize P, Z.
$EP = \Phi$
$w^1, w^2, ..., w^N$ are obtained using Algorithm 1.
while $t < gen\_$ max **do**
  $t = t + 1$
  for $i = 1, 2, ..., N$ **do**
    $B(i) = (i_1, i_2, ..., i_T)$ is adaptively updated by Algorithm 2.
    k and l are randomly selected from $B(i)$, and a new solution, y, is produced based on x^k and x^l.
    A new solution, y', is generated using binary crossover and polynomial mutation method.
    for $j = 1, 2, ..., m$ **do**
      if $z_j > f_j(y')$ **then**
        Let $z_j \leftarrow f_j(y')$.
      end if
    end for
    for $j \in B(i)$ **do**
      if $g^{PBI}(y'|\lambda^j) \leq g^{PBI}(x^j|\lambda^j)$ **then**
        Let $x^j = y'$, and update the corresponding solution.
      end if
      Remove solutions dominated by $F(y')$ from EP.
    end for
  end for
  $i = i + 1$
end while

۱. کپی کردن (duplicating): دو عدد صحیح ، مثلا k و l، به صورت تصادفی انتخاب می‌شوند از مجموعه $B(i) = (i_1, i_2, \dots, i_T)$ و سپس یک راه حل جدید y با توجه به x^k, x^l ساخته می‌شود.

۲. پیشرفت کردن (improving): یک راه حل جدید y' توسط کراس اوور باینری و جهش چند جمله‌ای انجام می‌گیرد.

۳. بروزرسانی (updating): اگر $z_j > f_j(y')$ باشد در نتیجه $z_j \leftarrow f_j(y'), j = 1,2, \dots, m$ می‌نویسیم.

۴. بروزرسانی راه حل‌ها در همسایگی‌ها: اگر $y' \prec x^j, j \in B(i)$ باشد در نتیجه $x^j = y'$ و راه حل مورد نظر را بروز رسانی می‌کنیم.

۵. بروزرسانی EP: راه حل هایی که توسط $F(y')$ مغلوب می‌شوند را حذف می‌کنیم.

زمانی که به حداکثر تعداد نسل و تکرار رسیدیم الگوریتم را متوقف می‌کنیم و سپس EP را به عنوان خروجی الگوریتم می‌دهیم.

۵. طراحی تجربی

۵.۱ موارد آزمایش

برای آزمایش کردن الگوریتم از بهینه سازی های بنچمارک استفاده شده است:

ZDT	DTLZ	WFG	UF	MOTSPs
1~4,6	1~7	1~8	1~10	-

۵.۲ تنظیمات طراحی

آزمایشات به صورت دو گروه انجام شده است.گروه اول تاثیر تعداد نسل‌ها زمانی‌که بردار های وزن توسط مسئله DTLZ1 حل می‌شود محاسبه شده است. همچنین تاثیر پارامتر های مربوط به فرصت تنظیم همسایگی توسط مسئله DTLZ2 بررسی شده است. دومین گروه تلاش دارند تا نتیجه عملکرد روش پیشنهادی را بررسی نمایند.

برای همه این روش ها، عملگر های ژنتیک یکسانی در نظر گرفته شده است. به صورت تخصصی کراس اوور باینری و جهش چندجمله‌ای در نظر گرفته شده است و توزیع این دو عملگر برابر ۲۰ و ۱۰ در نظر گرفته شده است.

برای تعیین تعداد جمعیت Popsize که برابر کوچک ترین عددی است که ۴ برابر آن بزرگتر از تعداد وزن هاست. برای NSGA-III مقدار Popsize برابر ۹۲، ۲۱۲ و ۱۵۶ برای m های ۳، ۵ و ۸ می باشد. برای سایر روش های بهینه سازی این مقدار برابر ۹۱، ۲۱۰ و ۱۵۶ برای m های ۳، ۵ و ۸ می‌باشد.

همه الگوریتم های تکاملی بعد از ۹۱۰۰۰، ۱۵۶۰۰۰ و ۲۱۰۰۰۰ اجرا شدن تابع عملکرد برای m های برابر ۳، ۵ و ۸ خاتمه می‌یابند. برای هر تابع تست، هر روش به صورت مستقل ۳۰ بار اجرا می‌شود و مقادیر همه توابع عملکرد ذخیره شده و مقدار میانگین و انحراف معیار آن محاسبه می‌شود.

همه آزمایشات روی کامپیوتر با مشخصات intel core I5-5200u CPU 2.20GHz و 4GB RAM اجرا شده است.

۵.۳ اندازه گیری عملکرد

معیار های اندازه گیری عملکرد شامل:

۱. توزیع: اندازه گیرنده تقریبی یکنواختی راه حل های بهینه سطح پرتو بدست آمده توسط الگوریتم
۲. همگرایی: اندازه گیرنده تقریبی درجه سطح پرتو بدست آمده و میزان حقیقی آن
۳. زمان اجرا: میزان پیچیدگی زمانی الگوریتم

برای اندازه گیری توزیع و همگرایی از معیار IGD در مقاله استفاده شده است.

$$\mathrm{IGD}(P) = \frac{1}{|P^*|} \sum_{z^* \in P^*} d(z^*, P)$$

متغیر P مقدار تقریبی راه حل بدست آمده توسط الگوریتم را نشان می‌دهد. مقدار P* یک مجموعه یکنواختی صحیح رو PF را نشان می‌دهد. d(z*, P) کمترین فاصله اقلیدسی بین $z^* \epsilon P^*$ و P را نشان می‌دهد. همچنین |P*| میزان کاردینالیتی آن است. هرچقدر مقدار IGD الگوریتم کمتر باشد آن الگوریتم عملکرد بهتری دارد.

همچنین این مقاله از متریک HV نیز استفاده نموده است. نقاط رفرنس $r^* = (r_1^*, \ldots, r_m^*)$ در حقیقت نقاطی هستند که کمی بزرگتر از nadir points هستند. مقدار HV از مجموعه P حجم ناحیه ای است که توسط P با توجه به

$$\mathrm{HV}(P, r^*) = L\left(\bigcup_{x \in P} [F_1(x), r_1] \times \ldots \times [F_m(x), r_m]\right)$$

نقاط nadir مغلوب شده‌اند.

در این فرمول L(.) نشان دهنده اندازه لبگ یا Lebesgue measure می‌باشد.هرچقدر میزان HV بزرگتر باشد الگوریتم کارایی بهتری دارد.

۶. طراحی کد

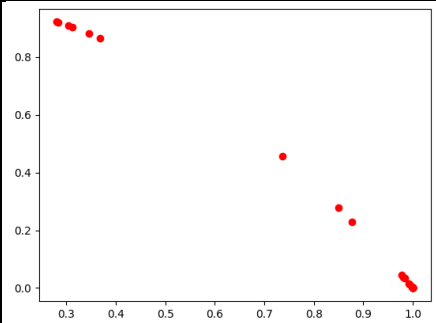
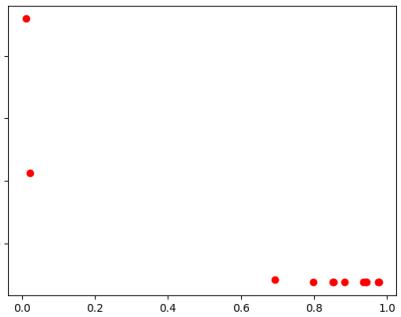
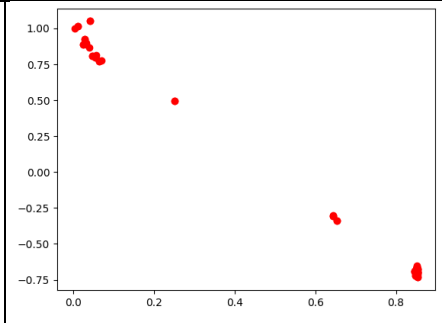
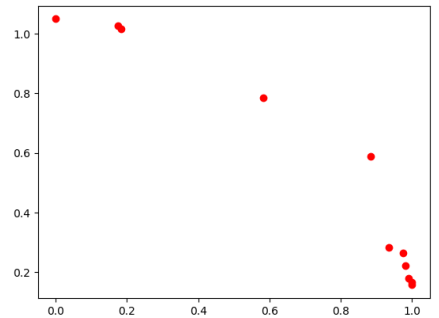
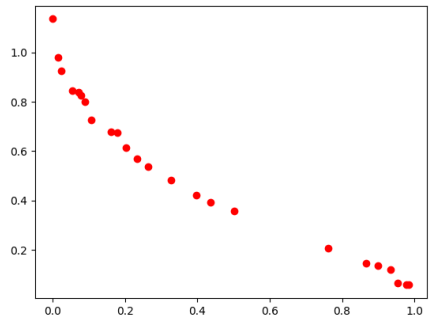
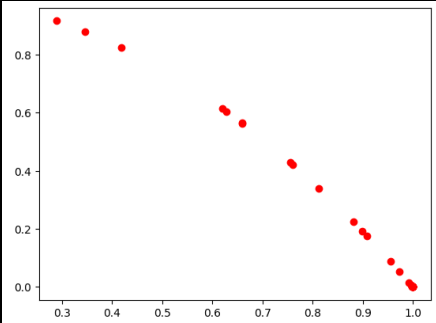
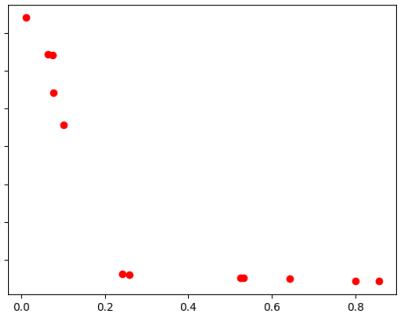
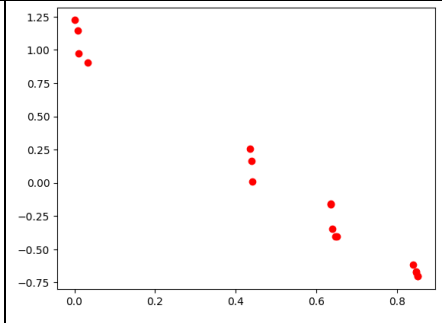
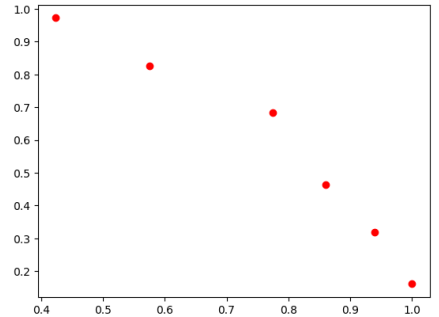
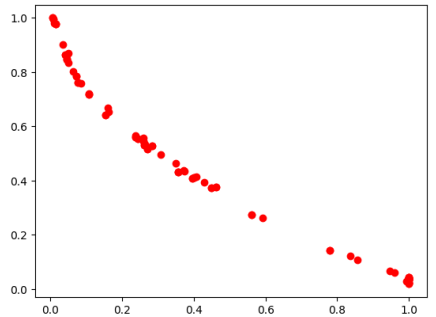
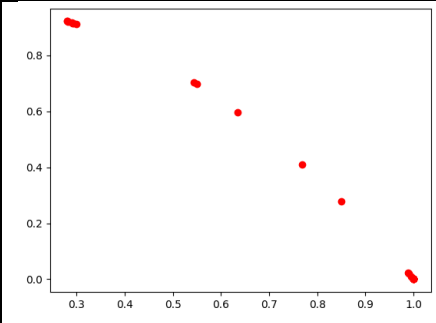
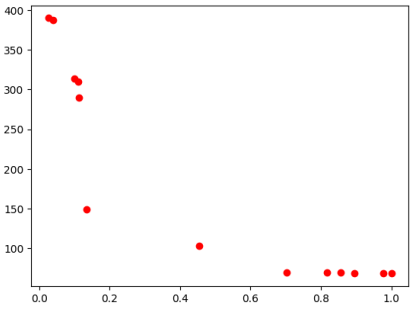
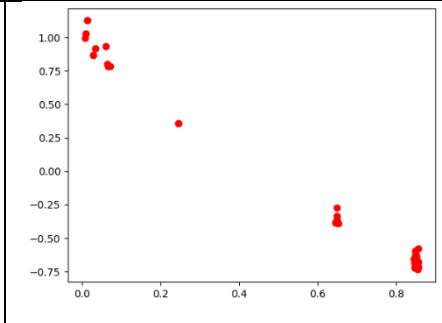
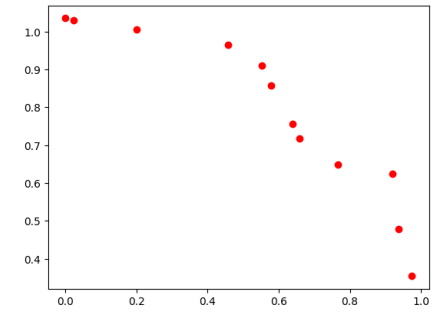
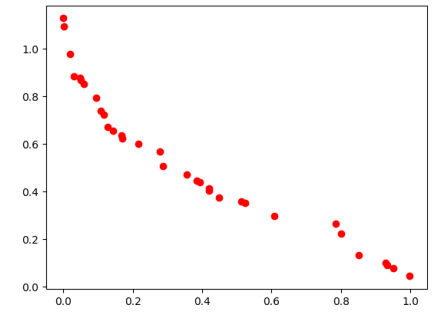
۶.۱ طراحی کلی کد

طراحی کلی کد شامل کلاس های زیر می‌باشد:

I. کلاس GeneticAlgorithm که این کلاس عملیات ساخت جمعیت اولیه، کراس اوور تک نقطه‌ای و جهش چند جمله‌ای را به نحوه خاص مقاله انجام داده است.

II. کلاس MOP که از کلاس اول ارث بری کرده است و عملیات های محاسبه هر تابع هدف، پیاده سازی توابع weighted_sum و PBI و محاسبه معیار ارزیابی IGD را امکان پذیر ساخته است.

III. کلاس MOEAD که از کلاس دوم ارث بری کرده است. این کلاس محاسبه Sparcity را برای الگوریتم اول ممکن ساخته است و از طرفی دارای تابع moead می‌باشد که باعث اجرا الگوریتم اصلی می‌شود. این کلاس شامل توابعی جهت پیدا کردن اهداف مغلوب و غالب است که کمک می‌کند جواب های مغلوب از خروجی حذف شوند. همچنین شامل تابع plot است که با دادن هرجمعیتی شروع به رسم نقشه خروجی های آن می‌کند.

ZDT6	ZDT4	ZDT3	ZDT2	ZDT1
				
				
				

جدول شماره (۱)

۶.۲ اجرا و مقایسه کد

جهت اجرای کد مربوطه از محیط colab استفاده شده است. همانطور که در جدول شماره (۱) مشاهده می‌کنید برای هر کدام از توابع بنچمارک معرفی شده در این مقاله برای چندین بار اجرا شده و اجرای سه بار از آنها در جدول قرار گرفته است.

با توجه به اشکال رسم شده در مقاله کد من در dense بودن اشکال دچار اشکال است که آن هم به دلیل optimize نبودن کافی کد است.

حال به سراغ مقایسه مقادیر IGD می‌رویم:

IGD	MOEAD AWA IGD	MY CODE
ZDT1	4.61e-3	2.86e-2
ZDT2	1.43e-1	1.117e-1
ZDT3	2.35e-2	2.28e-2
ZDT4	2.16e-1	-
ZDT6	1.29e-2	1.69e-2

با توجه به اینکه نسخه های متعددی از سایر توابع بنچمارک در توابع پایتون دیده می‌شود، تصمیم بر آن شد که کد فقط روی ۵ تابع تست گردد و مقادیر IGD آن با الگوریتم مقاله اصلی بررسی گردد. نکته قابل توجه آن است که کد من در تابع ZDT4 به درستی اجر نمی‌شود و اختلاف زیادی با سطح پرتو اصلی ایجاد می‌کند.

۶.۳ نوآوری

برای نوشتن کد مقاله مجبور به استفاده از روش های جدید جهت جلوگیری از سربار اضافی سیستم شدم.

به طور مثال برای محاسبه همسایگی برای بردار های وزن، از روش Nearest Neighbors یادگیری ماشین استفاده کردم تا با سرعت بیشتری بتوانم همسایگی ها را محاسبه کنم.

همچنین در هر بار حلقه به جای استفاده از تنها یک فرزند از هر دو فرزند استفاده کردم و در صورت بهتر بودن آنان را جایگزین والدین کردم.

همچنین برای جلوگیری از حلقه های پیچیده محاسباتی بخش دوم الگوریتم دوم را کمی تغییر دادم تا پیچیدگی محاسباتی کاهش پیدا کند.

۷. نتیجه گیری

این مقاله قسمت های نامعلوم زیادی دارد که راجع به آنها صحبتی نشده است. من جمله gene_max یا میزان نرخ جهش. از طرفی بخشی از این مقاله مختص به زمان اجرای الگوریتم های مختلف است که چون نوع سیستم مشخصی برای زمان بندی مشخص شده است نمی‌توان مقایسه درستی به عمل آورد. کد های توضیح داده شده در الگوریتم ها دارای حلقه های زیادی هستند که روند کلی حل مسئله را کند می‌کند. این مقاله راجع به مقدار دهی اولیه بردار های وزن نیز صحبتی نکرده است. همچنین محاسبه نقاط رفرنس نیز مبحث مهمی است که صحبتی به میان آورده نشده است. اما با توجه به مسائل گفته شده می‌توان از معیار IGD و شکل های رسم شد به دقت خوبی کد من با مقاله را بررسی نمود.