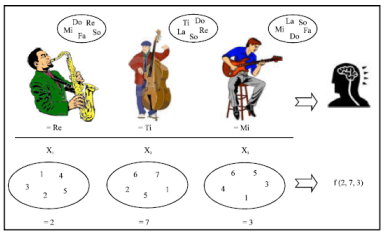
دانشگاه اصفهان

دانشکده مهندسی کامپیوتر – گروه هوش مصنوعی و رباتیک

گزارش تمرین سوم – بهینه‌سازی چندهدفه

سیستم توصیه‌گر

HAS + MA



پدیدآورنده:

محمد امین کیانی

۴۰۴۳۶۴۴۰۰۸

دانشجوی کارشناسی ارشد، دانشکده‌ی کامپیوتر، دانشگاه اصفهان، اصفهان،
Aminkianiworkeng@gmail.com

استاد درس: دکتر کارشناس

نیمسال اول تحصیلی ۰۵-۱۴۰۴

فهرست مطالب

مستندات ۳

۰- مسئله و تحلیل کلی آن: ۳

۱- وظیفه اول: مطالعه و تحلیل الگوریتم‌ها ۳

الگوریتم جستوجوی هارمونی (HSA) ۳

الگوریتم ممتیک (MA) ۹

۲- وظیفه دوم: تعریف بهینه‌سازی چند هدفه ۱۷

نمادگذاری و داده‌های ورودی ۱۷

متغیر تصمیم و فضای جستجو ۱۸

توابع هدف ۱۹

قید حداقل پوشش ژانری ۲۰

نرمال‌سازی اهداف ۲۳

مدیریت چندهدفه(ترکیب/انتخاب راه‌حل‌ها) ۲۳

(Bias / Fairness / Novelty) - اهداف اضافی پیشنهادی ۲٤

۳- مراجع ۲۸

مستندات

۰- مسئله و تحلیل کلی آن:

در اینجا به بررسی دو الگوریتم **جستجوی هارمونی** (**Harmony Search Algorithm – HSA**) و **ممتیک** (**Memetic Algorithm – MA**) می‌پردازیم. هر دو الگوریتم از نوع فراابتکاری هستند که برای حل مسائل بهینه‌سازی به کار می‌روند. ابتدا ماهیت هر الگوریتم را توضیح داده و سپس اجزا و عملگرهای اصلی آن‌ها را معرفی می‌کنیم و در ادامه نیز شبه‌کد نسخه استاندارد تک‌هدفه‌ی هر کدام ارائه شده است. در فازهای بعد به پیاده‌سازی و تحلیل نتایج الگوریتم‌ها پرداخته و درنهایت مقایسه‌ی گروهی برای پایان این روال صورت می‌گیرد.

۱- وظیفه اول: مطالعه و تحلیل الگوریتم‌ها

الگوریتم جستوجوی هارمونی (HSA)

ماهیت الگوریتم: الگوریتم جستجوی هارمونی یک روش بهینه‌سازی تکاملی الهام‌گرفته از روند بداهه‌نوازی موسیقی است. این الگوریتم نخستین بار توسط Geem و همکاران در سال ۲۰۰۱ ارائه شد و به دلیل سادگی مفهومی، تعداد کم پارامترها و عدم نیاز به مشتق یا گرادیان، به یکی از روش‌های پرکاربرد بهینه‌سازی تبدیل شده است. ایده کلی HSA تشبیه فرایند پیدا کردن ترکیب‌های موسیقیایی هماهنگ (هارمونیک) به فرایند جستجوی راه‌حل‌های بهینه در مسائل ریاضی است. به جای تأکید بر استعاره موسیقی، در اینجا بر مکانیزم عملی الگوریتم تمرکز می‌کنیم : HSA با نگهداری مجموعه‌ای از راه‌حل‌ها (که **حافظه هارمونی** نامیده می‌شود) تلاش می‌کند ترکیبی بهینه از تصمیم‌ها را با استفاده از **انتخاب از حافظه، تنظیم (جهش) جزئی مقادیر** و **کاوش تصادفی** بیابد.(HSA از اطلاعات تمام جمعیت برای تولید فرزند جدید بهره می‌برد که نوعی "بازترکیب جهانی" (Global Recombination) محسوب می‌شود.)

اجزا و عملگرهای اصلی در HSA : الگوریتم جستجوی هارمونی دارای مولفه‌ها و پارامترهای کلیدی زیر است که رفتار جستجو را تعیین می‌کنند:

- **حافظه هارمونی(Harmony Memory – HM)** : مجموعه‌ای از راه‌حل‌های فعلی (بردارهای تصمیم) است که الگوریتم آن‌ها را حفظ می‌کند. اندازه این حافظه با پارامتری به نام HMS (Harmony Memory Size)مشخص می‌شود. هر راه‌حل را می‌توان به یک قطعه موسیقی (یک بردار هارمونیک) تشبیه کرد. الگوریتم با ترکیب اطلاعات راه‌حل‌های موجود در HM راه‌حل‌های جدید می‌سازد.

- **نرخ ملاحظه حافظه(Harmony Memory Consideration Rate – HMCR)** : احتمالی بین ۰ و ۱ که تعیین می‌کند هر متغیر تصمیم در راه‌حل جدید چگونه انتخاب شود. با احتمالHMCR ، مقدار یک متغیر جدید **از بین مقادیر موجود در حافظه هارمونی** انتخاب می‌شود (استفاده از دانش راه‌حل‌های قبلی) و یک مقدار **کاملاً تصادفی** برای آن متغیر از فضای جستجو انتخاب خواهد شد. این مکانیزم نوعی **انتخاب مبتنی بر حافظه** است که تضمین می‌کند الگوریتم از تجربیات گذشته استفاده کند.

- **نرخ تنظیم گام (Pitch Adjustment Rate – PAR) و پهنای باند(Bandwidth – bw)** : پس از انتخاب یک مقدار از حافظه برای متغیر (طبقHMCR)، با احتمال PAR آن مقدار دستخوش یک تغییر جزئی می‌شود. این تغییر به اندازه کوچکی (مثلاً اضافه/کم‌کردن یک مقدار دلتا) صورت می‌گیرد که میزان حداکثر تغییر توسط پارامتر **bw** تعیین می‌شود. به این عملگر اصطلاحاً **تنظیم گام** می‌گویند که مشابه عمل جهش (Mutation) در الگوریتم‌های تکاملی است . PAR معمولاً یک مقدار کوچک (مثلاً ۰.۱ یا ۰.۲) در نظر گرفته می‌شود تا تنظیمات جزئی موجب **جستجوی محلی** پیرامون راه‌حل‌های موجود شود. ترکیب PAR و bw امکان کاوش دقیق‌تر در حوالی راه‌حل‌های فعلی را فراهم می‌کند.

- **انتخاب تصادفی:** زمانی که انتخاب از حافظه انجام نمی‌شود، مقدار یک متغیر به صورت کاملاً تصادفی از دامنه ممکن متغیر برگزیده می‌شود. این مؤلفه نقشی شبیه **کاوش جهانی**

(Exploration) را ایفا می‌کند و تنوع جمعیت را بالا نگه می‌دارد تا الگوریتم در بهینه‌های محلی گرفتار نشود. در واقع تعادل بین استفاده از حافظه(استفاده از دانش کسب‌شده، استثمار یا Exploitation) و انتخاب تصادفی (کاوش موارد جدید، Exploration) از طریق تنظیم HMCR برقرار می‌شود.

- **تابع هدف (Objective Function) و ارزیابی:** هر راه‌حل (هارمونی) از طریق یک تابع شایستگی یا هدف ارزیابی می‌شود. الگوریتم تک‌هدفه HSA به دنبال بیشینه‌سازی یا کمینه‌سازی این تابع هدف برای پیدا کردن بهترین راه‌حل است. مقدار این تابع برای هر هارمونی تعیین‌کننده کیفیت آن راه‌حل است.

- **به‌روزرسانی حافظه:** حافظه هارمونی همواره فقط تعداد محدودی راه‌حل (HMS عدد) را نگهداری می‌کند. پس از تولید هر راه‌حل جدید و محاسبه شایستگی آن، اگر این راه‌حل از بدترین راه‌حل حاضر در HM بهتر باشد، جایگزین آن می‌شود. این مکانیزم مشابه **بقا و انتخاب طبیعی** است که تضمین می‌کند کیفیت کلی حافظه در طول زمان بهبود یابد. به عبارت دیگر، HM همیشه تمایل دارد بهترین راه‌حل‌های یافت‌شده تاکنون را حفظ کند.

- **شرط توقف:** الگوریتم جستجوی هارمونی تا زمانی که شرط توقف برقرار نشده باشد، به تولید راه‌حل جدید و بروزرسانی حافظه ادامه می‌دهد. شروط توقف رایج شامل رسیدن به تعداد معینی تکرار/ارزیابی، یافتن راه‌حلی با کیفیت مطلوب یا عدم بهبود قابل‌توجه طی چندین تکرار اخیر است.

شبه‌کد نسخه تک‌هدفه استاندارد HSA : روند کلی الگوریتم جستجوی هارمونی در حالت بهینه‌سازی تک‌هدفه به صورت زیر خلاصه می‌شود:

الگوریتم جستجوی هارمونی (تک‌هدفه)

---

۱. تنظیم پارامترهای اولیه:

- تعیین اندازه حافظه هارمونی (HMS)
- تعیین HMCR, PAR, bw و سایر پارامترهای مسئله (مانند دامنه متغیرها)
- تعریف تابع هدف برای ارزیابی کیفیت راه‌حل‌ها

۲. مقداردهی اولیه حافظه هارمونی (HM):

- تولید HMS راه‌حل اولیه به صورت تصادفی
- ارزیابی هر راه‌حل و ذخیره در حافظه هارمونی

۳. تکرار فرایند بدست آوردن هارمونی جدید تا زمان ارضای شرط توقف:

الف. **تولید هارمونی جدید**:

برای هر متغیر تصمیم *i* در بردار راه‌حل:

- تولید یک عدد تصادفی یکنواخت در بازه [0,1].
- اگر عدد تصادفی < HMCR:

مقدار *x_i* را از بین مقادیر قبلی موجود در ستون *i*ام حافظه هارمونی انتخاب کن (انتخاب از HM).

در غیر این صورت:

مقدار *x_i* را به صورت تصادفی از دامنه معتبر متغیر انتخاب کن.

- اگر مقدار *x_i* از حافظه انتخاب شده بود:

یک عدد تصادفی جدید در [0,1] تولید کن.

اگر این عدد < PAR:

مقدار *x_i* را با یک تغییر جزئی تنظیم کن (افزایش یا کاهش کوچک حداکثر به اندازه bw).

ب. **ارزیابی هارمونی جدید**:

مقدار تابع هدف را برای بردار تصمیم جدید محاسبه کن.

پ. **به‌روزرسانی حافظه هارمونی**:

اگر شایستگی (کیفیت) هارمونی جدید بهتر از بدترین هارمونی فعلی در HM باشد:

- هارمونی جدید را جایگزین بدترین هارمونی در حافظه کن (حافظه را به‌روز کن).

ت. **ادامه حلقه**:

مراحل تولید تا به‌روزرسانی را تا زمانی که شرط پایان (مثلاً تعداد تکرار معین یا رسیدن به بهینگی مطلوب) برآورده شود تکرار کن.

۴. خاتمه الگوریتم:

- به عنوان خروجی، بهترین راه‌حل موجود در حافظه هارمونی (یا مجموعه‌ای از بهترین‌ها) ارائه می‌شود.

1) Harmony Search Algorithm (HSA) — Single-Objective Standard Pseudocode

Assumptions / Inputs

- Objective function: `f(x)` (either **minimize** or **maximize**)
- Decision vector: `x = (x1, x2, ..., xN)` with domains `D1, D2, ..., DN`
- Parameters:
  - `HMS` = Harmony Memory Size (population size)
  - `HMCR` ∈ [0,1] = Harmony Memory Consideration Rate
  - `PAR` ∈ [0,1] = Pitch Adjustment Rate
  - `bw` > 0 = bandwidth (step size for pitch adjustment; used when meaningful)
  - `MaxIters` = maximum improvisation iterations
- Comparator:
  - `Better(a,b)` returns true if solution `a` is better than `b` (depends on min/max)

Helper Functions

- `RandomSample(Di)` → sample a valid value from domain `Di`
- `PitchAdjust(value, Di, bw)` → returns a *neighbor* value of `value` within `Di`
- `ArgWorst(HM)` → index of worst harmony in memory (by objective)
- `ArgBest(HM)` → index of best harmony in memory (by objective)

Pitch Adjustment (generic)

You must implement this depending on variable type:

- **Continuous:** `value' = clamp(value + Uniform(-bw, +bw), Di)`
- **Integer:** `value' = clamp(value + RandomChoice({-1,+1}) * step, Di)` (e.g., `step=1` or derived from `bw`)
- **Categorical / permutation:** apply a small neighborhood move (swap, insert, replace, etc.)

```
Algorithm HSA_SingleObjective(f, {Di}i=1..N, HMS, HMCR, PAR, bw, MaxIters):

    # ----------------------------
    # 1) Initialize Harmony Memory
    # ----------------------------
    HM = empty list
    for j = 1 to HMS:
        x = empty vector of length N
        for i = 1 to N:
            x[i] = RandomSample(Di)
        x.f = f(x)
        HM.add(x)
```

```
    # (Optional) keep HM sorted by quality to speed up best/worst access

    best = HM[ArgBest(HM)]

    # ----------------------------
    # 2) Main improvisation loop
    # ----------------------------
    for iter = 1 to MaxIters:
        # 2.a) Improvise a new harmony x_new
        x_new = empty vector length N

        for i = 1 to N:
            r = Uniform(0, 1)

            if r < HMCR:
                # Memory consideration: pick a value from existing HM for dimension i
                # Option A: pick a random harmony and copy its i-th variable
                h = RandomInteger(1, HMS)
                x_new[i] = HM[h][i]

                # Pitch adjustment with probability PAR
                r2 = Uniform(0, 1)
                if r2 < PAR:
                    x_new[i] = PitchAdjust(x_new[i], Di, bw)
            else:
                # Random consideration: sample from domain
                x_new[i] = RandomSample(Di)

        # 2.b) Evaluate
        x_new.f = f(x_new)

        # 2.c) Update Harmony Memory (replace worst if new is better)
        w = ArgWorst(HM)
        if Better(x_new, HM[w]):
            HM[w] = x_new

        # 2.d) Track best
        b = ArgBest(HM)
        if Better(HM[b], best):
            best = HM[b]

    # ----------------------------
    # 3) Return best solution found
    # ----------------------------
    return best
```

در شبه‌کد فوق، مشاهده می‌شود که الگوریتم HSA چگونه با ترکیب سه گزینه (انتخاب از حافظه، تنظیم مقداری کوچک، یا انتخاب کاملاً تصادفی) یک راه‌حل جدید می‌سازد. این روش سبب می‌شود که هم **تنوع** راه‌حل‌ها حفظ شود (از طریق انتخاب‌های تصادفی) و هم از **تجربیات** قبلی استفاده بهینه شود (از طریق حافظه هارمونی). در نسخه تک‌هدفه، معیار "بهترین" بودن هارمونی بر اساس مقدار عددی تابع هدف است؛ یعنی الگوریتم یک راه‌حل بهینه‌ی نهایی را بازمی‌گرداند. در مسائل چندهدفه (که در وظایف بعدی مد نظر است)، به جای بهترین تک‌راه‌حل، مجموعه‌ای از راه‌حل‌های غیرأست dominated (جبهه‌ی پارتو) مدنظر خواهد بود که در ادامه به آن پرداخته می‌شود.

الگوریتم ممتیک (MA)

ماهیت الگوریتم: الگوریتم ممتیک یک روش بهینه‌سازی تکاملی **ترکیبی** است که با افزودن جستجوی محلی به رویه‌های یک الگوریتم ژنتیک (GA) استاندارد، کارایی جستجو را بهبود می‌دهد. نام "ممتیک" از مفهوم میم (واحد انتقال فرهنگی یا ایده) گرفته شده که توسط ریچارد داوکینز معرفی شد و در اینجا به این معناست که هر راه‌حل (هر فرد در جمعیت) می‌تواند مستقل از دیگران تکامل محلی پیدا کند. به عبارت دیگر، MAرا می‌توان **الگوریتم ژنتیک به همراه بهبود موضعی** در نظر گرفت؛ از این رو به آن **الگوریتم تکاملی هیبریدی** نیز می‌گویند. این الگوریتم تلاش می‌کند **مزایای جستجوی سراسری** تکاملی (که تنوع زیادی ایجاد می‌کند) را با **مزایای جستجوی محلی** (که به صورت موضعی راه‌حل را بهبود می‌بخشد) ترکیب کند تا به کیفیت بهتری در حل مسئله برسد. در ادامه اجزای اصلی MA را بررسی می‌کنیم.

اجزا و عملگرهای اصلی در MA : الگوریتم ممتیک شامل تمامی مولفه‌های یک الگوریتم ژنتیک استاندارد به‌علاوه یک مولفه‌ی جستجوی محلی است:

- **نمایش راه‌حل و جمعیت اولیه:** مانند سایر الگوریتم‌های تکاملی، ابتدا باید نحوه نمایش (کدگذاری) هر راه‌حل مشخص شود (مثلاً یک بردار اعداد حقیقی، یک رشته باینری، یک دنباله ترتیب آیتم‌ها و غیره). سپس یک **جمعیت اولیه** از راه‌حل‌ها (معمولاً به صورت

تصادفی) ایجاد می‌شود. اندازه جمعیت یک پارامتر مهم است. هر عضو جمعیت را می‌توان مشابه یک "فرد" یا یک کاندید پاسخ در نظر گرفت.

- **تابع هدف و ارزیابی:** یک تابع شایستگی (هدف) برای سنجش کیفیت هر فرد تعریف می‌شود. در حالت تک‌هدفه، این تابع ممکن است امتیازی برای مطلوبیت راه‌حل باشد که الگوریتم سعی در بیشینه کردن (یا کمینه کردن) آن دارد. تمامی افراد جمعیت اولیه پس از ایجاد، توسط تابع هدف ارزیابی و مقدار شایستگی‌شان مشخص می‌شود.
- **انتخاب والدین(Selection)** : در هر نسل، تعدادی از افراد به عنوان **والدین** برای تولید فرزندان جدید انتخاب می‌شوند. روش‌های مختلفی برای انتخاب وجود دارد (همانند الگوریتم ژنتیک)، از جمله **چرخ رولت** (متناسب با شایستگی)، **تورنمنت** (بین چند انتخاب تصادفی، بهترین برگزیده می‌شود) و یا **انتخاب رتبه‌بندی‌شده**. هدف مرحله انتخاب این است که به راه‌حل‌های بهتر شانس بیشتری برای تولید نسل بعدی بدهیم، در عین حال تنوع ژنتیکی نیز حفظ شود.
- **ترکیب(تقاطع یا Crossover)**: عملگر ترکیب روی والدین انتخاب‌شده اعمال می‌شود تا **فرزندان** جدید ایجاد شوند. این عملگر بخش‌هایی از اطلاعات دو (یا چند) والد را با هم ترکیب می‌کند. برای مثال، در نمایش برداری ممکن است یک نقطه‌ی برش تعیین شود و بخش ابتدایی بردار از والد اول و بخش انتهایی از والد دوم گرفته شود (یک‌نقطه‌ای یا دونقطه‌ای)، یا در نمایش‌های دیگر، روش‌های ترکیب خاصی تعریف می‌گردد. نتیجه، تولید راه‌حل‌های جدیدی است که ترکیبی از خصوصیات والدین هستند. نرخ ترکیب معمولاً یک پارامتر بین ۰ و ۱ است (مثلاً ۰.۸) که تعیین می‌کند با چه احتمالی یک جفت والدین واقعاً ترکیب شوند.
- **جهش(Mutation)** : پس از ایجاد فرزندان توسط ترکیب، یک عملگر جهش روی فرزندان اعمال می‌شود تا تغییرات تصادفی کوچکی در آن‌ها ایجاد کند. جهش تضمین می‌کند که تنوع در جمعیت حفظ شده و الگوریتم در یک مسیر تکاملی محدود گیر نیفتد. میزان و نوع

جهش وابسته به کدگذاری راه‌حل است (مثلاً تغییر تصادفی یک بیت در رشته باینری، یا تغییر اندک یک مقدار در بردار حقیقی، یا جابجایی دو عنصر در یک دنباله). نرخ جهش نیز یک پارامتر الگوریتم (معمولاً کوچک، مثلاً ۰.۰۱ یا ۰.۰۵) است که احتمال اعمال جهش روی هر فرد یا هر ژن را تعیین می‌کند.

- **جستجوی محلی(Local Search)** : این مولفه وجه تمایز اصلی الگوریتم ممتیک از یک GA ساده است. پس از ایجاد هر فرزند (یا پس از انتخاب برخی از فرزندان)، یک رویه‌ی بهبود موضعی روی آن فرد اجرا می‌شود. جستجوی محلی می‌تواند هر الگوریتم بهینه‌سازی تک‌نقطه‌ای باشد که از پاسخ فعلی شروع شده و سعی می‌کند یک بهبود کوچک در آن ایجاد کند. برای مثال، اگر مسئله پیوسته باشد می‌توان یک گام گرادیانی کوچک (در صورت محاسبه‌پذیر بودن) یا تغییرات تصادفی محدود در همسایگی انجام داد؛ یا اگر مسئله گسسته باشد می‌توان از عملگرهای تبادل یا حرکت‌های کوچک در فضای همسایگی بهره برد. این مرحله تضمین می‌کند که فرزندان تولیدشده فوراً **بهبود کیفی محلی** پیدا کنند و تا حد امکان در ناحیه پیرامون خود بهینه‌تر شوند. معمولاً اعمال جستجوی محلی برای همه فرزندان ممکن است زمان‌بر باشد، بنابراین گاهی فقط روی بهترین فرزندان یا به صورت دوره‌ای انجام می‌شود. انتخاب نوع و شدت جستجوی محلی یکی از مباحث طراحی در MA است که باید با فرایند تکاملی کلی متعادل شود (اگر جستجوی محلی بیش از حد قوی باشد، ممکن است تنوع را کاهش داده و الگوریتم در بهینه محلی گیر بیفتد).

- **جایگزینی (Replacement) و به‌روزرسانی جمعیت:** پس از ایجاد و بهبود فرزندان، کل جمعیت باید برای نسل بعد به‌روزرسانی شود. در ساده‌ترین حالت (الگوریتم ژنتیک سنتی) کل والدین با فرزندان جدید **جایگزین** می‌شوند (نسل جدید کاملاً جانشین نسل قبل می‌گردد). اما در بسیاری از موارد، به‌ویژه درMA ، از استراتژی‌های پیشرفته‌تر استفاده می‌شود تا اطلاعات مفید از نسل قبل حفظ شود؛ برای مثال ممکن است **نخبه‌گرایی** (Elitism) به کار رود که طی آن یکی چند تا از بهترین راه‌حل‌های نسل قبل مستقیماً به نسل بعد منتقل می‌شوند تا از بین نروند. یا ممکن است تعداد افراد جمعیت ثابت نگه داشته

شود و ترکیبی از بهترین والدین و فرزندان، جمعیت جدید را تشکیل دهند. هدف این مرحله آن است که جمعیت نسل بعد همچنان دارای اندازه ثابت باشد و آماده تکرار فرآیند تکاملی در گام بعدی شود.

- **شرط توقف:** مشابه دیگر الگوریتم‌های تکاملی، حلقه تولید نسل‌ها در MA نیز تا زمانی ادامه می‌یابد که شرط(های) پایان ارضا شوند. شرط توقف می‌تواند رسیدن به تعداد مشخصی نسل/ارزیابی، رسیدن به سطح مطلوب تابع هدف، یا عدم بهبود محسوس طی چند نسل اخیر باشد. پس از توقف، به‌طور معمول بهترین فرد یافت‌شده یا کل جمعیت نهایی (برای تحلیل) خروجی داده می‌شود.

شبه‌کد نسخه تک‌هدفه استاندارد MA: الگوریتم ممتیک در حالت تک‌هدفه (مثلاً بیشینه‌سازی یک تابع معین) را می‌توان به صورت زیر خلاصه کرد:

```
الگوریتم ممتیک (تک‌هدفه)
---------------------------
۱. مقداردهی اولیه:
- تعریف نحوه‌ی نمایش راه‌حل (کدگذاری کروموزوم‌ها)
- تنظیم پارامترها (اندازه جمعیت N، نرخ ترکیب pc، نرخ جهش pm و ...)
- تولید جمعیت اولیه شامل N راه‌حل (تصادفی یا با استفاده از دانش مسئله)
- ارزیابی اولیه جمعیت (محاسبه تابع هدف برای هر فرد)

۲. حلقه تکاملی (تا زمانی که شرط توقف برقرار نباشد):
الف. **انتخاب والدین:**
- با توجه به مقادیر شایستگی، تعدادی والد از جمعیت جاری انتخاب کن (مثلاً به روش تورنمنت یا چرخ رولت).
ب. **تولید فرزندان با ترکیب:**
- والدین انتخاب‌شده را جفت کن و با احتمال pc عملگر ترکیب را روی هر جفت اعمال کن تا کروموزوم‌های فرزند تولید شوند.
پ. **اعمال جهش:**
- روی هر کروموزوم فرزند با احتمال pm جهش (تغییر تصادفی کوچک) اعمال کن.
ت. **جستجوی محلی روی فرزندان:**
- برای هر فرزند (یا برخی از فرزندان منتخب)، رویه بهبود محلی را اعمال کن تا کروموزوم اندکی بهینه‌تر شود.
- ارزیابی مجدد فرزندان پس از جستجوی محلی (محاسبه تابع هدف به‌روز شده).
ث. **تشکیل نسل جدید (جایگزینی):**
- ترکیب والدین و فرزندان یا فقط استفاده از فرزندان برای به‌روزرسانی جمعیت:
* (حالت ساده) نسل جدید = تعداد N فرزند تازه ایجادشده (نسل قبلی کاملاً جایگزین می‌شود).
* (حالت نخبه‌گرا) چند فرد برتر از نسل قبل + بقیه از بهترین فرزندان = نسل جدید.
- در هر صورت، اندازه جمعیت را به N فرد در نسل جدید تنظیم کن.
ج. **به‌روزرسانی بهترین راه‌حل پیداشده (اختیاری):**
- اگر لازم است، بهترین فرد فعلی را در جایی ذخیره کن (برای گزارش یا استفاده آتی).

۳. پایان الگوریتم:
- خروجی: بهترین راه‌حل یا مجموعه‌ای از بهترین راه‌حل‌های به‌دست‌آمده طی تکامل.
```

2) Memetic Algorithm (MA) — Single-Objective Standard Pseudocode

A Memetic Algorithm = **Evolutionary Algorithm (e.g., GA)** + **Local Search** (improvement) on individuals (often offspring).

Assumptions / Inputs

- Objective function: `f(x)` (minimize or maximize)
- Representation: defines what an individual `x` looks like (binary, real vector, permutation, ranked list, ...)
- Parameters:
  - `PopSize` = population size `N`
  - `pc` $\in$ [0,1] = crossover probability
  - `pm` $\in$ [0,1] = mutation probability (per individual or per gene, depending on your design)
  - `MaxGens` = number of generations
  - `EliteSize` = number of elites carried over (elitism)
  - `pLS` $\in$ [0,1] = probability of applying local search to an offspring (common design)
  - `LS_Budget` = maximum local search steps (or evaluations) per LS call
- Operators:
  - `InitializeIndividual()` → creates a random feasible solution
  - `SelectParent(P)` → e.g., tournament selection
  - `Crossover(p1, p2)` → returns children `(c1, c2)` (type depends on representation)
  - `Mutate(x, pm)` → mutated individual
  - `LocalSearch(x, f, LS_Budget)` → locally improved individual
- Comparator:
  - `Better(a,b)` like above

A Generic Local Search Template :

```
Procedure LocalSearch_HillClimb(x, f, LS_Budget):
    x_best = x
    f_best = f(x_best)

    for step = 1 to LS_Budget:
        x_n = GenerateNeighbor(x_best)     # small move in neighborhood
        f_n = f(x_n)

        if Better( (x_n,f_n), (x_best,f_best) ):
            x_best = x_n
            f_best = f_n
```

```
    return x_best with fitness f_best
```

MA Pseudocode (Generational + Elitism + LS on offspring):

```
Algorithm MA_SingleObjective(f, PopSize, pc, pm, MaxGens, EliteSize, pLS,
LS_Budget):

    # ----------------------------
    # 1) Initialize population
    # ----------------------------
    P = empty list
    for i = 1 to PopSize:
        x = InitializeIndividual()
        x.f = f(x)
        P.add(x)

    best = BestIndividual(P)   # by Better()

    # ----------------------------
    # 2) Evolutionary loop
    # ----------------------------
    for gen = 1 to MaxGens:

        # 2.a) Create next generation
        P_next = empty list

        # --- Elitism: copy top EliteSize individuals
        elites = TopK(P, EliteSize)    # sorted by quality
        P_next.add_all(elites)

        # --- Generate offspring until population is full
        while |P_next| < PopSize:

            # Selection
            p1 = SelectParent(P)
            p2 = SelectParent(P)

            # Crossover
            r = Uniform(0,1)
            if r < pc:
                (c1, c2) = Crossover(p1, p2)
            else:
                c1 = Clone(p1)
```

```
            c2 = Clone(p2)

        # Mutation
        c1 = Mutate(c1, pm)
        c2 = Mutate(c2, pm)

        # Local Search (Memetic part)
        r1 = Uniform(0,1)
        if r1 < pLS:
            c1 = LocalSearch(c1, f, LS_Budget)
        else:
            c1.f = f(c1)

        r2 = Uniform(0,1)
        if r2 < pLS:
            c2 = LocalSearch(c2, f, LS_Budget)
        else:
            c2.f = f(c2)

        # Add offspring (respect PopSize)
        if |P_next| < PopSize:
            P_next.add(c1)
        if |P_next| < PopSize:
            P_next.add(c2)

    # 2.b) Replace population
    P = P_next

    # 2.c) Update global best
    current_best = BestIndividual(P)
    if Better(current_best, best):
        best = current_best

    # (Optional) early stopping if no improvement for T generations, etc.

# ----------------------------
# 3) Return best found
# ----------------------------
return best
```

در این شبه‌کد، مشاهده می‌شود که MA همان ساختار کلی الگوریتم ژنتیک را دنبال می‌کند (مراحل انتخاب، ترکیب، جهش، جایگزینی)، با این تفاوت مهم که مرحله‌ی ت. جستجوی محلی روی فرزندان اضافه شده است. این مرحله اجازه می‌دهد هر راه‌حل به صورت فردی بهبود یابد؛ به

عبارتی هر "میم" (راه‌حل) می‌تواند مستقل یاد بگیرد و بهتر شود. نتیجه این است که الگوریتم معمولاً با سرعت همگرایی بیشتری به راه‌حل‌های خوب می‌رسد، زیرا کاوش سراسری GA برای نقاط امیدوارکننده از طریق جستجوی محلی تثبیت و تقویت می‌شود. البته باید توجه داشت که هزینه محاسباتی MA معمولاً بیشتر از GA معمولی است (به دلیل صرف زمان برای جستجوی محلی در هر نسل)، بنابراین یکی از ملاحظات مهم، تعادل بین شدت جستجوی محلی و اندازه/تعداد نسل‌های تکاملی است.

به صورت خلاصه:

- **در HSA:** جستجو از طریق ترکیب ویژگی‌های موجود در حافظه انجام می‌شود. برای مسئله Top-K این یعنی الگوریتم سعی می‌کند فیلم‌هایی را که در لیست‌های موفق قبلی بوده‌اند، در ترکیب‌های جدید امتحان کند.

```
- Algorithm 1: Standard Harmony Search (Single-Objective)
- 1. Initialize Parameters: HMS, HMCR, PAR, MaxIterations.
- 2. Initialize HM: Generate HMS random solutions in the candidate space.
- 3. Evaluate: Calculate fitness f(x) for all solutions in HM.
- 4. While (t < MaxIterations):
-     a. Improvise a new solution x_new = (x_1, x_2,..., x_K):
-        For each dimension j = 1 to K:
-            If rand(0,1) < HMCR:
-                x_j = value from HM[random_index][j]
-                If rand(0,1) < PAR:
-                    x_j = Adjust(x_j) // Small change or neighbor swap
-            Else:
-                x_j = random_item_from_candidates()
-     b. Selection/Update:
-        If f(x_new) is better than f(x_worst) in HM:
-            Replace x_worst with x_new.
-     c. t = t + 1.
- 5. Return: Best solution in HM.
```

- **در MA** : تمرکز بر این است که ابتدا یک ترکیب کلی خوب از آیتم‌ها پیدا شود (تقاطع) و سپس با جابجایی‌های کوچک (جستجوی محلی)، دقت یا تنوع آن لیست به حداکثر برسد. در رویکرد **لامارکی** (رایج در سیستم‌های توصیه‌گر)، هم مقدار برازندگی و هم ساختار راه حل (ژن‌ها) پس از جستجوی محلی به‌روزرسانی می‌شوند.

```
- Algorithm 2: Standard Memetic Algorithm (Single-Objective)
- 1. Initialize: Create an initial population of size N.
- 2. Evaluate: Calculate fitness f(x) for each individual.
- 3. While (Termination Condition not met):
-     a. Selection: Choose parents from the current population.
-     b. Recombination: Apply Crossover to parents to generate offspring
  P_off.
-     c. Mutation: Apply Mutation to P_off to generate P_mut.
-     d. Local Search (The Memetic Step):
-        For each individual x in P_mut:
-            x_refined = Local_Search_Procedure(x)
-            // e.g., greedy swaps to improve fitness
-            x = x_refined
-     e. Evaluation: Calculate f(x) for the refined individuals.
-     f. Survival: Select N individuals for the next generation (e.g.,
  elitism).
- 4. Return: Best solution found.
```

۲- وظیفه دوم: تعریف بهینه‌سازی چند هدفه

در این بخش، مسئله انتخاب فهرست Top-K به عنوان یک مسئله بهینه‌سازی ترکیبیاتی با اهداف متضاد و قیود مشخص تعریف می‌شود.(برای هر دو الگوریتم)

نمادگذاری و داده‌های ورودی

- $\mathcal{U}$:مجموعه کاربران
- $\mathcal{I}$:مجموعه آیتم‌ها (فیلم‌ها)

$u \in \mathcal{U}$ برای هر کاربر ، اطلاعات زیر در اختیار است :

۱) **مجموعه آیتم‌های کاندید** (آیتم‌هایی که کاربر قبلاً تعامل/مشاهده نداشته است) :

$$\mathcal{C}_u \subseteq \mathcal{I}$$

۲) **امتیاز پیش‌بینی‌شده‌ی مدل برای زوج** (u, i):

$$\hat{r}_{u,i} \in \mathbb{R}$$

۳) **اطلاعات ژانری هر آیتم** به‌صورت یک بردار باینری :

$$\mathbf{g}_i \in \{0,1\}^G$$

متغیر تصمیم و فضای جستجو

برای هر کاربر یک راه‌حل، یک **فهرست مرتب** از طول K است :

$$\mathbf{x} = (i_1, i_2, \dots, i_K)$$

با شرایط زیر :

۲) هر آیتم از کاندیدها انتخاب شود :

$$i_j \in \mathcal{C}_u \quad \forall j = 1, \dots, K$$

۳) آیتم‌ها تکراری نباشند :

$$i_a \neq i_b \quad \forall a \neq b$$

۴) ترتیب آیتم‌ها مهم است (رتبه‌بندی حفظ می‌شود) .

بنابراین فضای جستجو :

$$\Omega_u = \{(i_1, \dots, i_K) \mid i_j \in \mathcal{C}_u,\ i_a \neq i_b\}$$

توابع هدف

برای هر فهرست پیشنهادی برای کاربر، اهداف اصلی عبارت‌اند از :

- بیشینه‌سازی دقت/مرتبط‌بودن (Accuracy / Precision proxy)
- بیشینه‌سازی تنوع درون‌فهرستی (Intra-List Diversity)

هدف اول: دقت/مرتبط‌بودن

میانگین امتیاز پیش‌بینی‌شده (ساده و رایج) :

$$f_1(\mathbf{x}; u) = \frac{1}{K}\sum_{j=1}^{K} \hat{r}_{u,i_j}$$

هدف: $\max f_1(\mathbf{x}; u)$

معیار رتبه‌محور (پیشنهادی چون ترتیب مهم است) :

ابتدا مفهوم زیر را تعریف می‌کنیم :

$$\mathrm{DCG}(\mathbf{x}; u) = \sum_{j=1}^{K} \frac{\hat{r}_{u,i_j}}{\log_2(j+1)}$$

سپس لیست ایده‌آلی با مرتب‌سازی کاندیدها بر اساس امتیاز ساخته می‌شود و :

$$\mathrm{IDCG}(u) = \mathrm{DCG}(\mathbf{x}^{\star}; u)$$

در نهایت :

$$f_1^{\mathrm{NDCG}}(\mathbf{x}; u) = \frac{\mathrm{DCG}(\mathbf{x}; u)}{\mathrm{IDCG}(u) + \varepsilon}$$

$\varepsilon > 0$ برای جلوگیری از تقسیم بر صفر است که هدف $\max f_1^{\mathrm{NDCG}}(\mathbf{x}; u)$ است .

هدف دوم: تنوع درون‌فهرستی با فاصله Jaccard

برای هر آیتم مجموعه ژانرهای آن را تعریف می‌کنیم :

$$\Gamma(i) = \{t \in \{1, \dots, G\} \mid g_{i,t} = 1\}$$

شباهت جاکارد بین دو آیتم :

$$\text{JaccardSim}(i,j) = \frac{|\Gamma(i) \cap \Gamma(j)|}{|\Gamma(i) \cup \Gamma(j)| + \varepsilon}$$

فاصله (عدم‌شباهت) جاکارد :

$$d_J(i,j) = 1 - \text{JaccardSim}(i,j)$$

تنوع درون‌فهرستی :

$$\text{ILD}(\mathbf{x}) = \frac{2}{K(K-1)} \sum_{1 \le p < q \le K} d_J\left(i_p, i_q\right)$$

پس :

$$f_2(\mathbf{x}) = \text{ILD}(\mathbf{x})$$

هدف $\max f_2(\mathbf{x})$ است.

قید حداقل پوشش ژانری

برای جلوگیری از راه‌حل‌های بدیهی (مثلاً انتخاب آیتم‌های بسیار مشابه از یک ژانر)، قید زیر اعمال می‌شود .

ژانرهای پوشش داده‌شده توسط لیست :

$$\Gamma(\mathbf{x}) = \bigcup_{j=1}^{K} \Gamma\left(i_j\right)$$

تعداد ژانرهای متمایز در لیست :

$$\mathrm{GC}(\mathbf{x}) = |\Gamma(\mathbf{x})|$$

قید حداقل پوشش :

$$\mathrm{GC}(\mathbf{x}) \geq M$$

M یک مقدار ثابت (هایپرپارامتر) است.

روش اعمال قید

برای مدیریت قید حداقل پوشش چند روش استاندارد وجود دارد. باید یکی را انتخاب و شفاف گزارش‌کنیم.

ابتدا مقدار «نقض قید» را تعریف می‌کنیم :

$$v(\mathbf{x}) = \max(0,\ M - \mathrm{GC}(\mathbf{x}))$$

روش (تعمیر راه‌حل)

اگر قید برقرار نبود، با جایگزینی تعدادی آیتم کاری می‌کنیم که ژانرهای جدید وارد لیست شوند ، تا شرط برقرار گردد .

ایده‌ی اجرایی: تا زمانی که‌قید برقرار نیست، یک ژانرِ پوشش‌داده‌نشده انتخاب می‌شود و آیتمی که آن ژانر را دارد (و در لیست نیست) جایگزین یکی از آیتم‌های کم‌اثر (مثلاً آیتم انتهای لیست یا آیتم با امتیاز کمترین) می‌شود .

روش (تابع جریمه)

در صورت استفاده از ترکیب وزن‌دار اهداف، یک تابع تک‌هدفه جریمه‌دار می‌سازیم :

$$J(\mathbf{x}) = w_1 \tilde{f}_1(\mathbf{x}) + w_2 \tilde{f}_2(\mathbf{x}) - \lambda \tilde{v}(\mathbf{x})$$

که در آن :

$$\tilde{v}(\mathbf{x}) = \frac{v(\mathbf{x})}{M}$$

و $w_1, w_2 \geq 0$ و $w_1 + w_2 = 1$ و $\lambda > 0$.

روش (انتخاب مبتنی بر شدنی بودن)

در مقایسه‌ی دو راه‌حل :

۱) اگر فقط یکی شدنی باشد، پس بهتر است .

۲) اگر هر دو شدنی باشند، با معیار چندهدفه (پارتو یا وزن‌دار) مقایسه می‌شوند .

۳) اگر هر دو نشدنی باشند، آنکه نقض کمتری دارد بهتر است :

$v(\mathbf{x}) < v(\mathbf{y}) \Rightarrow \mathbf{x}$ بهتر از $\mathbf{y}$

روش (عملگرهای آگاه از قید)

در طراحی جهش/ترکیب/حرکت، انتخاب‌هایی انجام می‌دهیم که احتمال تولید لیست‌های ناقض قید کم شود؛ مثلاً هنگام جایگزینی آیتم، ترجیح دهیم آیتم جدید ژانرِ جدید اضافه کند .

نرمال‌سازی اهداف

برای مقایسه منصفانه و همچنین ترکیب اهداف (در روش وزن‌دار) لازم است اهداف نرمال شوند .

Min–Max روی جمعیت/حافظه

برای هدف(f) روی مجموعه مرجع (P) داریم :(Harmony Memory یا جمعیت)

$$f_m^{\min} = \min_{\mathbf{x}\in\mathcal{P}} f_m(\mathbf{x}), \qquad f_m^{\max} = \max_{\mathbf{x}\in\mathcal{P}} f_m(\mathbf{x})$$

نرمال‌سازی :

$$\tilde{f}_m(\mathbf{x}) = \frac{f_m(\mathbf{x}) - f_m^{\min}}{f_m^{\max} - f_m^{\min} + \varepsilon}$$

$\tilde{f}_m(\mathbf{x}) \in [0,1]$.

نرمال‌سازی ذاتی

اگر از هدف معیار رتبه‌محور استفاده شود، خودِ هدف دقت ذاتاً در بازه ۰ تا ۱قرار می‌گیرد و همچنین تنوع درون فهرستی‌نیز معمولاً در بازه‌ی ۰ تا ۱ است .

مدیریت چندهدفه(ترکیب/انتخاب راه‌حل‌ها)

سه راهبرد استاندارد برای چندهدفه کردن مسئله وجود دارد .

غلبه پارتو - پیشنهاد اصلی

برای بیشینه‌سازی، می‌گوییم یکی بر دیگری غالب است اگر :

$$\forall m:\ f_m(\mathbf{x}) \geq f_m(\mathbf{y}) \quad \text{و} \quad \exists m:\ f_m(\mathbf{x}) > f_m(\mathbf{y})$$

خروجی طبیعی این روش، مجموعه‌ی راه‌حل‌های **غیرمغلوب** (جبهه پارتو) است. برای حفظ تنوع روی جبهه می‌توان از معیارهایی استفاده کرد (مثل crowding distance).

ترکیب وزن‌دار

یک تابع تک‌هدفه از اهداف نرمال‌شده ساخته می‌شود:

$$J(\mathbf{x}) = \sum_{m=1}^{M} w_m \, \tilde{f}_m(\mathbf{x})$$

$$w_m \geq 0 \quad . \sum_m w_m = 1$$

روش قید ε

یکی از اهداف را بهینه و بقیه را قید می‌کنیم. مثال :

$$\max \; f_1(\mathbf{x}; u) \quad \text{s.t.} \quad f_2(\mathbf{x}) \geq \varepsilon, \; \mathrm{GC}(\mathbf{x}) \geq M$$

با تغییر اپسیلون نقاط مختلف trade-off تولید می‌شوند .

(Bias / Fairness / Novelty) - اهداف اضافی پیشنهادی

به‌جز دقت و تنوع، اهداف دیگری نیز می‌توان اضافه کرد. هدف‌های زیر در سیستم‌های توصیه‌گر بسیار رایج و قابل دفاع هستند .

هدف اضافه ۱: کاهش بایاس محبوبیت / افزایش Novelty

محبوبیت یک آیتم در داده‌ی آموزش :

$$\mathrm{pop}(i) = |\{(u, i) \in \mathcal{D}_{train}\}|$$

(الف) کمینه‌سازی جریمه محبوبیت

$$f_3^{\text{pop}}(\mathbf{x}) = \frac{1}{K}\sum_{j=1}^{K} \log\left(1 + \text{pop}(i_j)\right)$$

هدف : $\min f_3^{\text{pop}}$

(ب) بیشینه‌سازی Novelty

$$f_3^{\text{nov}}(\mathbf{x}) = \frac{1}{K}\sum_{j=1}^{K} [\,-\log(1 + \text{pop}(i_j))]$$

هدف : $\max f_3^{\text{nov}}$

هدف اضافه ۲: عدالت/کالیبراسیون ژانری

برای اینکه توزیع ژانری توصیه‌ها خیلی نامتوازن نباشد یا با سلیقه‌ی کاربر هم‌راستا شود ، می‌توان فاصله‌ی بین توزیع‌ها را کمینه کرد .

ابتدا وزن رتبه به سبک DCG :

$$w_j = \frac{1}{\log_2(j+1)}$$

نمایش ژانر در لیست :

$$E_t(\mathbf{x}) = \sum_{j=1}^{K} w_j \; g_{i_j,t}$$

توزیع ژانری لیست :

$$q_{\mathbf{x}}(t) = \frac{E_t(\mathbf{x})}{\sum_{\ell=1}^{G} E_\ell(\mathbf{x}) + \varepsilon}$$

(الف) Fair Exposure | نسبت به توزیع هدف یکنواخت

$$p_{\text{target}}(t) = \frac{1}{G}$$

(ب) Calibration | نسبت به پروفایل ژانری کاربر

اگر تاریخچه کاربر در داده‌ی آموزش :

$$\mathcal{H}_u = \{i \mid (u,i) \in \mathcal{D}_{train}\}$$

توزیع ژانری کاربر :

$$p_u(t) = \frac{\sum_{i \in \mathcal{H}_u} g_{i,t}}{\sum_{\ell=1}^{G} \sum_{i \in \mathcal{H}_u} g_{i,\ell} + \varepsilon}$$

فاصله Jensen–Shannon :

ابتدا :

$$m(t) = \frac{1}{2}(q_{\mathbf{x}}(t) + p(t))$$

سپس :

$$D_{\mathrm{JS}}(q_{\mathbf{x}} \parallel p) = \frac{1}{2}\sum_{t=1}^{G} q_{\mathbf{x}}(t)\log\frac{q_{\mathbf{x}}(t)}{m(t)} + \frac{1}{2}\sum_{t=1}^{G} p(t)\log\frac{p(t)}{m(t)}$$

می‌توان هدف را به یکی از دو شکل تعریف کرد :

- کمینه‌سازی فاصله :

$$\min\ D_{\mathrm{JS}}(q_{\mathbf{x}} \parallel p)$$

یا

- بیشینه‌سازی نسخه منفی فاصله :

$$f_4(\mathbf{x}; u) = -D_{\mathrm{JS}}(q_{\mathbf{x}} \parallel p)$$

پس صورت نهایی مسئله چندهدفه برای هر کاربر :

$$\max_{\mathbf{x}\in\Omega_u} \mathbf{F}(\mathbf{x}; u) = \max_{\mathbf{x}\in\Omega_u} (f_1(\mathbf{x}; u),\ f_2(\mathbf{x}),\ f_3(\mathbf{x}),\ f_4(\mathbf{x}; u))$$

به‌طوری‌که :

$$\mathrm{GC}(\mathbf{x}) \geq M$$

نکته‌ی اجرایی برای استفاده در MA و HSA :

- در هر دو الگوریتم، یک راه‌حل برابر یک لیست است .
- برای هر راه‌حل باید بردار اهداف محاسبه شود .
- قید پوشش ژانری باید با یکی از روش‌های گفته‌شده، اعمال شود .
- برای چندهدفه کردن، پیشنهاد اصلی استفاده از **پارتو** است؛ اما روش وزن‌دار نیز قابل استفاده است .

٣- مراجع

[1] https://github.com
[2] https://stackoverflow.com/questions
[3] https://colab.research.google.com/
[4] https://www.researchgate.net/publication/348194344
[5] https://chatgpt.com/
[6] https://gemini.google.com/