# ۱ مدل‌سازی وظایف و سامانه

## ۱.۱ تولید گراف DAG

تابع generate_dag سه خانوادهٔ کلاسیک گراف را پوشش می‌دهد. در این بخش، هر مدل را به‌تفصیل شرح می‌دهیم.

### ۱.۱.۱ مدل Erdős–Rényi

- **تعریف:** در مدل $\mathrm{ER}(n,p)$ هر یک از $\binom{n}{2}$ جفت رأس با احتمال مستقل $p$ به‌هم متصل می‌شوند.
- **پارامترها:** $\langle n, p\rangle$؛ در پیاده‌سازی حاضر، $p$ به‌صورت $\min\{0.5, \frac{3}{n}\}$ تنظیم می‌شود مگر آن‌که کاربر مقدار دیگری بدهد.
- **ویژگی‌ها:**
  - توزیع درجه تقریباً دوجمله‌ای (در حد بزرگ‌نمونه، پواسن).
  - فاقد ساختار خوشه‌ای مشخص؛ قطر گراف حول $\log n$ است.
  - آستانه‌های فاز مشخص برای ظهور مولفهٔ غول‌آسا و هم‌بندی.
- **کاربردها:** خط مبنا برای مقایسهٔ شبکه‌های پیچیده؛ مناسب برای شبیه‌سازی «بی‌نظم کامل».
- **مزایا/معایب:** سادگی پارامتر و پیاده‌سازی؛ اما عدم انطباق با توزیع درجات ناهمسانِ شبکه‌های واقعی.

### ۱.۱.۲ مدل مقیاس‌آزاد Barabási–Albert

- **تعریف:** رشد تدریجی + ترجیح اتصال؛ در هر گام یک رأس تازه با $m$ یال وارد می‌شود و با احتمالی متناسب با درجهٔ فعلیِ رؤوس موجود، به آن‌ها متصل می‌گردد.
- **پارامترها:** $\langle n, m\rangle$. مقدار $m$ پیش‌فرض ۲ است.
- **ویژگی‌ها:**
  - توزیع درجه از نوع Power-law با نمای $-3$.
  - وجود «ابرگره»هایی با درجهٔ بسیار بالا $\Rightarrow$ آسیب‌پذیری نقطه‌ای.
  - قطر کوچک ($\sim \log n$) ولی وابسته به مراکز پردرجه.
- **کاربردها:** مدل‌سازی شبکه‌های وب، تعاملات اجتماعی و پروتئینی که در آن «برندگان بزرگ‌تر می‌شوند».
- **مزایا/معایب:** بازتاب دقیق توزیع درجاتِ واقعیت؛ در عوض خوشه‌بندی پایین و فاکتور تجمعی کمتر از شبکه‌های دنیای واقعی.

### ۱.۱.۳ مدل دنیای کوچک Watts–Strogatz

- **تعریف:** آغاز از یک حلقهٔ مرتب $k$ـمنتظم، سپس بازسیمی (rewire) هر یال با احتمال $p$ به یک رأس تصادفی.
- **پارامترها:** $\langle n, k, p\rangle$. در کد حاضر $k = 4$ و $p = 0.1$ پیش‌فرض هستند.
- **ویژگی‌ها:**
  - خوشه‌بندی بالا (شبیه شبکهٔ منسجم محلی).
  - طول مسیر میانگین کوتاه ($\sim \log n$) به‌واسطهٔ یال‌های دوربرد.

– توزیع درجهٔ نسبتاً باریک—برخلاف مقیاس‌آزاد.

- **کاربردها:** سیستم‌های اجتماعی، نورونی و شبکه‌های لجستیکی که در آن‌ها «آشنای مشترک» فراوان ولی «شش درجه جدایی» نیز برقرار است.
- **مزایا/معایب:** تعادل خوشه‌بندی و قطر کوتاه؛ اما درجات ناهمسان واقعی را بازتولید نمی‌کند.

پس از تولیدِ هر گرافِ بدون‌جهت، یال‌ها بر پایهٔ رتبهٔ تصادفی رأس‌ها جهت‌دار می‌شوند تا یک DAG قطعی حاصل شود.

## ۲.۱ ساخت سامانه

```
def build_system(num_cores=4, heterogeneity=False, seed=None):
    ...
```

اگر heterogeneity=True باشد سرعت هر هسته به‌طور تصادفی در بازهٔ ۰.۵–۱.۵ تنظیم می‌شود؛ در غیر این‌صورت، تمام هسته‌ها سرعت ۱ دارند.

هر گره به شیئی از نوع Task با فیلدهای «workload»، «deadline» و resources تبدیل می‌شود. مهلت با ضریب deadline_factor (پیش‌فرض ۳) ضرب‌در حجمِ کار و کمی نویز تصادفی محاسبه می‌شود.

## ۳.۱ ساختار گراف وظایف

پس از تولید گراف بدون‌جهت با یکی از سه مدل بالا، فرآیند زیر برای ساخت گراف جهت‌دار انجام می‌شود:

- به هر رأس یک مقدار تصادفی نسبت داده شده و ترتیب آن‌ها به‌عنوان رتبه در نظر گرفته می‌شود.
- به‌ازای هر یال $(u, v)$، اگر $\text{rank}(u) < \text{rank}(v)$، یال جهت‌دار $(u \to v)$ حفظ می‌شود، در غیر این‌صورت حذف می‌گردد.
- در صورت نبود مسیر میان‌همهٔ گره‌ها، تنها قوی‌ترین مؤلفهٔ متصل نگه داشته می‌شود.

## ۴.۱ مدل بار کاری و مهلت‌ها

حجم کاری workload هر گره از توزیع نرمال مثبت یا توزیع یکنواخت $\text{Uniform}(1, 10)$ استخراج می‌شود.

- **ضریب مهلت‌دهی:** مقدار پیش‌فرض ۳ است، یعنی $deadline = 3 \times workload + \epsilon$ که $\epsilon \sim \text{Uniform}(-0.5, +0.5)$ است.
- این باعث می‌شود بیشتر وظایف قابل زمان‌بندی باشند اما برخی چالش‌برانگیز باقی بمانند.

## ۵.۱ پیاده‌سازی شیءگرا

هر گره با استفاده از کلاس Task مدل‌سازی می‌شود:

```
class Task:
    def __init__(self, id, workload, deadline, resources):
        ...
```

در صورتی‌که سیستم ناهمگن باشد، resources برای هر گره تصادفی تخصیص می‌یابد و محدودیت تخصیص روی هسته‌ها اعمال می‌شود.

# ۲ رابط خط فرمان (CLI)

نمونهٔ فراخوانی:

```
python scheduler.py --nodes 20 --model scale_free \
                    --cores 8 --hetero --seed 42 \
                    --out-dag dag.json --out-sched sched.json
```

پس از اجرا:

- فایل‌های dag.json و schedule.json تولید می‌شوند.
- روی خروجی استاندارد، تعداد وظایف و طول زمان‌بندی کلی (Makespan) چاپ می‌شود.

## ۱.۲ خروجی زمان‌بندی

فایل schedule.json ساختاری مانند زیر دارد:

```
[
  {
    "task_id": 51,
    "core_id": 0,
    "start": 0.0,
    "finish": 13.16451396465417
  }
  ...
]
```

- start و end زمان‌های شروع و پایان نسبی اجرای هر وظیفه هستند.
- core شمارهٔ هستهٔ تخصیص‌یافته را نشان می‌دهد.
- اجرای وظایف در چارچوب DAG و بدون نقض تقدم‌ها تضمین می‌شود.
- finish زمان اتمام تسک

## ۲.۲ ذخیره‌سازی DAG

فایل dag.json ساختار کلی زیر را دارد:

```
{
  {
      "id": 0,
      "workload": 50,
      "deadline": 140.283276704142,
      "resources": {
        "core": 1
      }
    },
  "edges": [
    {
      "source": 0,
      "target": 89
    },
}
```