Pipeline de simulación de grafos (feCUDA)

Este documento resume el flujo completo para generar grafos cuadrados sintéticos en C++, ejecutar el algoritmo de efectos y recolectar métricas de longitud de caminos para contrastar la hipótesis teórica.

Hipótesis a probar

- Se fija `epsilon > 1/2` para garantizar que los efectos (aristas con peso) se produzcan en un solo nivel por par `(i, j)`.
- Longitud máxima de caminos (`eta_0`):
  - **Sparse subcrítico** (grado medio < 1): `E[eta_0] = O(1)`.
  - **Denso** (grado medio > cN, c>1, o p constante): `E[eta_0] = O(1)`.
  - **Supercrítico sparse** (grado medio en (1, D)): `E[eta_0] = 0(log N)` con cotas `a log N < E[eta_0] < b log N`, para `a > b > 0`.

Módulos nuevos

- `include/utils/graph_generator.cuh` / `src/utils/graph_generator.cu`
  - Genera grafos Erdős–Rényi G(N, p) cuadrados (M == N) con pesos en `[epsilon, 1]` y simetría.
  - Regímenes: `sparse`, `supercritical`, `dense`.
  - Parámetros clave: `avg_degree`, `max_degree` (supercrítico), `dense_p` (denso), `epsilon`, `seed`.
  - Retorna estadísticas empíricas: `target_p`, `empirical_p`, `empirical_avg_degree`.
- `include/utils/eta_metrics.cuh` / `src/utils/eta_metrics.cu`
  - Calcula métricas de camino a partir de los tensores de rutas: `eta0_max`, `eta0_mean`, `eta0_p95`, `eta0_std`, `samples`.
  - Longitud aproximada: `cols - 2` por fila de camino (`[batch_id, n0, n1, ..., nk]`).
- Integración en `src/main.cu`

- Nuevas flags CLI: `--graph`, `--regime`, `--avg-degree`, `--max-degree`, `--dense-p`, `--epsilon`, `--graph-seed`, `--simulate-graphs`, `--scale-N`.
- Modo grafo usa el generador en lugar de datasets; valida `epsilon > 0.5` y `M == N`.
- Loguea métricas `ETA_STATS` por iteración.
- Escribe resultados en `results/graph_sim.csv` con parámetros, densidades y métricas.

Flujo del pipeline

1. **Configuración**: elegir régimen y parámetros que fijen el grado medio esperado.
2. **Generación de grafo** (host, C++):
   - Calcula `p` según régimen:
     - `sparse`: `p = avg_degree / (N-1)`, acotado a [0, 1].
     - `supercritical`: `p = clamp(avg_degree, 1, max_degree) / (N-1)`.
     - `dense`: `p = max(dense_p, avg_degree/(N-1))`, con mínimo suave 0.05.
   - Muestra aristas Bernoulli(p), asigna pesos uniformes en `[epsilon, 1]`, sin loops.
   - Retorna `TensorResult` en CPU y stats empíricas.
3. **Ejecución del algoritmo**:
   - Copia el tensor a GPU y llama `iterative_maxmin_cuadrado` (pipeline existente) para construir caminos.
   - Calcula métricas `eta_0` con `compute_eta_stats`.
4. **Registro de resultados**:
   - Consola: `GRAPH_GEN`, `ETA_STATS`, `BENCH_ITER`.
   - Archivo: `results/graph_sim.csv` (append) con parámetros y métricas por iteración.

Uso práctico (ejemplos)

Sparse subcrítico (esperar O(1))

```
./build/fecuda_main \
  --bench --synthetic --graph \
  --regime sparse --avg-degree 0.5 \
  --epsilon 0.6 \
  --batch 1 --M 512 --N 512 \
  --order 3 --threshold 0.2 \
  --replicas 1 --iterations 1
```

Supercrítico sparse (esperar Θ(log N))

```
./build/fecuda_main \
  --bench --synthetic --graph \
  --regime supercritical --avg-degree 2.0 --max-degree 4.0 \
  --epsilon 0.6 \
  --batch 1 --M 1024 --N 1024 \
  --order 3 --threshold 0.2 \
  --replicas 1 --iterations 1
```

Denso (esperar O(1))

```
./build/fecuda_main \
  --bench --synthetic --graph \
  --regime dense --dense-p 0.2 \
  --epsilon 0.6 \
  --batch 1 --M 256 --N 256 \
  --order 3 --threshold 0.2 \
  --replicas 1 --iterations 1
```

Barrido de tamaños (modo simulate)

```
./build/fecuda_main \
  --simulate-graphs --graph \
  --regime supercritical --avg-degree 2.0 --max-degree 4.0 \
  --epsilon 0.6 \
  --batch 1 --scale-N 256,512,1024,2048 \
  --order 3 --threshold 0.2 \
  --replicas 1 --iterations 1 \
  --label sweep_supercrit
```

Qué medir y cómo contrastar

- **Escalamiento**: graficar `eta0_mean` (o `eta0_max`) vs `log N` a partir de `results/graph_sim.csv`.
  - `sparse` y `dense` deben mostrar curvas planas (plafón O(1)).
  - `supercritical` debe crecer lineal en `log N`; la pendiente es candidato a las cotas `(a, b)`.
- **Regularidad de grado**: usar `empirical_avg_degree` y `empirical_p` para verificar que el régimen simulado se mantiene cerca del objetivo.
- **Sensibilidad a epsilon**: repetir con `epsilon` en (0.5, 0.9] para ver si cambian las pendientes o los plafones.

Notas prácticas

- Solo grafos cuadrados (`M == N`), y `epsilon` siempre debe ser `> 0.5`.
- El cálculo de longitud de camino usa `cols - 2` por fila; si se ajusta el formato de paths, actualizar `compute_eta_stats`.
- Los resultados se acumulan en `results/graph_sim.csv`; borrar/rotar manualmente si se desea empezar limpio.