# 精确介数中心性计算：文献回顾与 GPU 复现实验

## 1 问题定义

给定无权图 $G = (V, E)$。对任意 $s, t \in V$ 且 $s \neq t$，记 $\sigma_{st}$ 为从 $s$ 到 $t$ 的最短路径条数，$\sigma_{st}(v)$ 为这些路径中经过顶点 $v$ 的条数。则顶点 $v$ 的介数中心性 $\mathrm{BC}(v)$ 定义为:

$$\mathrm{BC}(v) = \sum_{s \neq t,\ s \neq v,\ t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

介数中心性刻画了顶点在全图最短路径上的中介（桥梁）作用，常用作衡量节点重要性的指标。本项目旨在对所有顶点 $v \in V$ 计算精确的 $\mathrm{BC}(v)$ 值，从而筛选关键节点。

实现上述目标的经典算法是 Brandes' algorithm。它对源点 $s$ 和目标点 $t$, 定义 $s \to t$ 对 $v$ 的依赖为

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}}$$

则源点 $s$ 对 $v$ 的总依赖为

$$\delta_s(v) = \sum_{t \in V \setminus \{s, v\}} \delta_{st}(v)$$

进而介数中心性 $\mathrm{BC}(v)$ 为

$$\mathrm{BC}(v) = \sum_{s \in V \setminus \{v\}} \delta_s(v)$$

在无权图上，Brandes 算法对每个源点 $s$ 执行一次 BFS，得到距离 $d_s(\cdot)$，最短路径条数 $\sigma_s(\cdot)$（其中 $\sigma_s(s) = 1$），并为每个顶点 $w$ 记录其前驱集合

$$P_s(w) = \{\, v \in V \mid (v, w) \in E,\ d_s(v) = d_s(w) - 1 \,\}$$

同时将顶点按 BFS 访问的非减距离顺序压入栈 $S$。随后初始化 $\delta_s(v) = 0$（对所有 $v \in V$），并按 $S$ 的出栈顺序（即距离非增）进行依赖回传：对每个弹出的顶点 $w$，对其每个前驱 $v \in P_s(w)$ 执行

$$\delta_s(v) \mathrel{+}= \frac{\sigma_s(v)}{\sigma_s(w)} \left(1 + \delta_s(w)\right)$$

最后对所有 $v \neq s$ 累加 $\mathrm{BC}(v) \mathrel{+}= \delta_s(v)$。总体而言，精确 BC 的计算需要对每个源点重复执行一次基于 BFS 的最短路遍历，并在其后进行依赖值的回传与累加。该过程对

边集的扫描与规约操作占据主导开销，使得无权图上的最坏时间复杂度达到 $O(|V||E|)$，在大规模图上往往难以承受。与此同时，尽管依赖回传存在层次顺序，但算法内部仍包含大量可并行的边访问、计数与累加（例如不同源点之间、以及同一层/同一顶点的前驱更新）。GPU 具备高吞吐并行与高带宽访存能力，天然适合加速此类“遍历 + 规约”主导的工作负载。因此，利用 GPU 加速精确 BC 的计算是一个具有潜力且价值明确的方向。

# 2 文献回顾

从计算模式上看，精确 BC 主要由多源 BFS（或 SSSP）、前沿扩展、以及依赖值回传过程中的大量规约与累加构成。这些算子与执行模式与 GPU 图分析引擎的核心抽象高度一致：图引擎通常围绕前沿（frontier）驱动的遍历与一系列可复用的并行原语（扫描、过滤、规约、原子更新、负载均衡与数据布局优化）来组织算法，从而能够较自然地承载并加速 BC 这类遍历主导的图指标计算。另一方面，围绕 GPU 图引擎的相关研究与工程实践已形成较为丰富的体系，为本项目提供了可复用的优化策略与实现参考。

基于上述原因，本章以 Gunrock 这一经典 GPU 图分析引擎为线索开展检索与归纳。表 1 汇总了已收集到的代表性论文，并记录其是否纳入本项目的主要参考范围以及未纳入的原因。

为了保证后续实现与对比的可复现性，本项目优先关注能够提供源码、并且在 GPU 上支持（或可扩展到支持）介数中心性计算的框架与算法工作。表中标记为“纳入”的条目将作为后续章节进一步讨论与实现的重点；其余条目则主要用于界定相关研究版图，并说明本项目为何不沿用这些思路（例如评测任务不包含 BC、计算平台不以 GPU 为主或研究方向偏离图分析引擎）。

表 1: 围绕 Gunrock 生态检索到的相关工作汇总

| 工作（会议/期刊，年份，被引） | 源码/链接 | 是否纳入 | 不纳入原因/备注 |
|---|---|---|---|
| Groute: An Asynchronous Multi-GPU Programming Model for Irregular Computations (ACM SIGPLAN Notices 52(8), 2017; 被引: 123) | `https://github.com/groute/groute` | 不纳入 | 文中评测任务不包含介数中心性（BC）。 |
| Hornet: An Efficient Data Structure for Dynamic Sparse Graphs and Matrices on GPUs (IEEE HPEC, 2018; 被引: 63) | `https://github.com/hornet-gt/hornet` | 不纳入 | 文中评测任务不包含介数中心性（BC）。 |
| Gunrock: GPU Graph Analytics (ACM TOPC, 2017; 被引: 119) | `https://github.com/gunrock/gunrock` | 纳入 | – |

| 工作（会议/期刊，年份，被引） | 源码/链接 | 是否纳入 | 不纳入原因/备注 |
| --- | --- | --- | --- |
| GraphBLAST: A High-Performance Linear Algebra-based Graph Framework on the GPU (ACM TOMS, 2022; 被引: 53) | https://github.com/gunrock/graphblast | 不纳入 | 文中评测任务不包含介数中心性（BC）。 |
| Tigr: Transforming Irregular Graphs for GPU-Friendly Graph Processing (ASPLOS, 2018; 被引: 106) | https://github.com/AutomataLab/Tigr | 纳入 | – |
| EGGPU: Enabling Efficient Large-Scale Network Analysis with Consumer-Grade GPUs (SocialMeta, 2024; 被引: 0) | https://github.com/easy-graph/Easy-Graph | 纳入 | – |
| Gunrock: A High-Performance Graph Processing Library on the GPU (PPoPP, 2016; 被引: 738) | https://github.com/gunrock/gunrock | 纳入 | – |
| Subway: Minimizing Data Transfer during Out-of-GPU-Memory Graph Processing (EuroSys, 2020; 被引: 105) | https://github.com/AutomataLab/Subway | 不纳入 | 文中评测任务不包含介数中心性（BC）。 |
| GraphIt: A High-Performance Graph DSL (OOPSLA, 2018; 被引: 250) | https://github.com/GraphIt-DSL/graphit | 不纳入 | 主要面向 CPU 的图处理系统，非 GPU-based。 |
| Gluon: a communication-optimizing substrate for distributed heterogeneous graph analytics (ACM SIGPLAN, 2018; 被引: 178) | – | 不纳入 | 文中评测任务不包含介数中心性（BC）。 |
| Fast Gunrock Subgraph Matching (GSM) on GPUs (arXiv, 2020; 被引: 26) | – | 不纳入 | 聚焦子图匹配任务，而非介数中心性（BC）。 |
| Computing Graph Neural Networks: A Survey from Algorithms to Accelerators (ACM Computing Surveys, 2023; 被引: 172) | – | 不纳入 | 综述文章，且研究方向与本项目不一致。 |
| HyGCN: A GCN Accelerator with Hybrid Architecture (HPCA, 2020; 被引: 284) | – | 不纳入 | 研究方向与本项目不一致。 |
| GraphR: Acceleration Graph Processing Using ReRAM (HPCA, 2018; 被引: 221) | – | 不纳入 | 文中评测任务不包含介数中心性（BC）。 |
| DataShot: Automatic Generation of Fact Sheets from Tabular Data (TVCG, 2020; 被引: 120) | – | 不纳入 | 研究方向与本项目不一致。 |

| 工作（会议/期刊，年份，被引） | 源码/链接 | 是否纳入 | 不纳入原因/备注 |
| --- | --- | --- | --- |
| P3: Distributed Deep Graph Learning at Scale (OSDI, 2021; 被引: 108) | – | 不纳入 | 研究方向与本项目不一致。 |
| Galliot: Path Merging Based Betweenness Centrality Algorithm on GPU (IEEE INFOCOM, 2023; 被引: 10) | – | 纳入 | – |

# 3 重点项目运行与复现

本章基于第二章表 1 中标记为“纳入”的相关工作，系统记录其编译、运行与复现流程，包括软硬件环境、代码版本、构建方式，以及 BC 任务的运行参数与输出口径等。上述信息用于保证后续实验对比的一致性，并为进一步的 GPU 加速实现沉淀可复用的工程基础。

根据第二章的筛选结果，本项目重点关注 Gunrock、EGGPU、Tigr 与 Galliot。其中，Gunrock、EGGPU 与 Tigr 均公开了源码。我们下载其代码并在服务器环境中完成编译与运行，以验证其在 BC 任务上的可复现性，并提取可借鉴的实现细节。需要说明的是，EGGPU（Easy-Graph）同时提供 CPU 版本（`easy_graph`）与可选的 GPU 模块；为启用 GPU 相关代码，编译时需在 `CMakeLists.txt` 中将 `EASYGRAPH_ENABLE_GPU` 设为 `ON`，否则默认仅构建 CPU 部分。

在复现过程中，我们发现 Tigr 的开源实现仅支持单源 BC（single-source BC），并未提供原生的全源（all-sources）计算接口；若需获得全图 BC，需要在外层对源点逐一迭代并汇总结果。另一方面，Gunrock 虽支持在命令行参数中传入多个源点，但当图规模较大时无法一次性传入全部源点，从而难以直接完成全源 BC。为此，我们对 Gunrock 的命令行参数解析代码（`include/gunrock/io/parameters.hxx`）进行了小幅修改：在原先仅支持形如 `--src 0,1,2,3` 的逗号分隔列表解析的基础上，增加对 `--src all` 的识别；当指定 `all` 时，将 $0, 1, \ldots, |V|-1$ 自动加入源点队列，从而支持全源 BC 的计算。对于 Galliot，由于其未公开源码，本项目主要参考论文中给出的算法思路与实验设置。

在完成编译与功能验证后，我们进一步在多个数据集上对 Gunrock 与 EGGPU 进行了性能复现。为提高实验可重复性，我们将运行流程固化为两个脚本 `run_eggpu.sh` 与 `run_gunrock.sh`。其中，EGGPU 通过 Python 调用编译后的 C/CUDA 动态库（`.so`）作为后端，因此我们使用 `run_easygraph_bc.py` 进行调用与参数配置。脚本与数据集路径以服务器目录 `/home/yuyuanhang/BC/` 为例，实际使用时可按环境调整。

`run_easygraph_bc.py`:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse
import os
import sys
import time

import easygraph as eg


def parse_sources(s: str):
    """
    支持:
      - "all"  -> None (表示全源)
      - "0"    -> [0]
      - "0,1,2"-> [0,1,2]
      - "@/path/to/sources.txt"  每行一个source id
    """
    if s is None or s.lower() == "all":
        return None
    if s.startswith("@"):
        path = s[1:]
        src = []
        with open(path, "r", encoding="utf-8") as f:
            for line in f:
                line = line.strip()
                if not line or line.startswith("#"):
                    continue
                src.append(int(line))
        return src
    return [int(x) for x in s.split(",") if x.strip() != ""]


def main():
    ap = argparse.ArgumentParser(
        description="Run EasyGraph/EGGPU betweenness centrality with configurable
    graph type and sources."
    )
    ap.add_argument("--input", required=True, help="Input edge list file. Format: u
     v [w]")
    ap.add_argument("--directed", action="store_true", help="Use DiGraph (directed)
    . Default: Graph (undirected)")
```

```
ap.add_argument("--weighted", action="store_true", help="Treat 3rd column as
weight")
ap.add_argument("--weight-key", default="weight", help="Edge weight key name (
default: weight)")
ap.add_argument("--sources", default="all",
                help='Sources for BC: "all" or "0" or "0,1,2" or "@/path/to/
sources.txt"')
ap.add_argument("--normalize", action="store_true", help="Normalize BC (if
supported by backend)")
ap.add_argument("--endpoints", action="store_true", help="Endpoints option (if
supported by backend)")
args = ap.parse_args()

if not os.path.exists(args.input):
    print(f"ERROR: input not found: {args.input}", file=sys.stderr)
    return 2

# 1) Build graph object
G = eg.DiGraph() if args.directed else eg.Graph()

# 2) Read edges into graph
t0 = time.time()
# EasyGraph 的 read_edgelist 支持 create_using，最稳妥：直接传实例（会清空后填
充）
# data 传入 weight tuple，会把第3列读入到 edge attr dict 里
if args.weighted:
    eg.read_edgelist(
        args.input,
        nodetype=int,
        data=((args.weight_key, float),),
        create_using=G,
    )
else:
    eg.read_edgelist(
        args.input,
        nodetype=int,
        data=False,
        create_using=G,
    )
t1 = time.time()

G = G.cpp()

# 3) Prepare sources
```

```
    sources = parse_sources(args.sources)

    # 4) Run betweenness centrality
    # EasyGraph 的 betweenness_centrality 支持 sources 参数（None=全源）
    # 注意：不同版本/后端对 normalize/endpoints 的支持可能不完全一致；
    # 如果你跑出 TypeError，就把对应参数关掉。
    t2 = time.time()
    bc = eg.betweenness_centrality(
        G,
        weight=(args.weight_key if args.weighted else None),
        sources=sources,
        normalized=args.normalize,
        endpoints=args.endpoints,
    )
    t3 = time.time()

    # 5) Output
    del bc

    return 0


if __name__ == "__main__":
    raise SystemExit(main())
```

run_eggpu.sh:

```
#!/usr/bin/env bash
set -euo pipefail

PY="/home/yuyuanhang/BC/repo/work/repos/easy_graph/run_easygraph_bc.py"
unset LD_LIBRARY_PATH

DATASETS_DIRECTED=(
  "/home/yuyuanhang/BC/repo/work/datasets/web-BerkStan.txt"
  "/home/yuyuanhang/BC/repo/work/datasets/cit-Patents.txt"
)

DATASETS_UNDIRECTED=(
  # "/home/yuyuanhang/BC/repo/work/datasets/com-orkut.txt"
)

run_one() {
```

```
  local mode="$1"
  local graph="$2"

  [[ -f "$graph" ]] || return 0

  if [[ "$mode" == "directed" ]]; then
    python3 "$PY" --input "$graph" --directed --sources all
  else
    python3 "$PY" --input "$graph" --sources all
  fi
}

for graph in "${DATASETS_DIRECTED[@]}"; do
  run_one directed "$graph"
done

for graph in "${DATASETS_UNDIRECTED[@]}"; do
  run_one undirected "$graph"
done
```

run_gunrock.sh:

```
#!/usr/bin/env bash
set -euo pipefail

BIN="/home/yuyuanhang/BC/repo/work/repos/gunrock/build/bin/bc"

# List of MTX files to run
MTX_FILES=(
  "/home/yuyuanhang/BC/repo/work/datasets/web-BerkStan.mtx"
  "/home/yuyuanhang/BC/repo/work/datasets/cit-Patents.mtx"
)

ok=0
fail=0

for MTX in "${MTX_FILES[@]}"; do
  if [[ ! -f "$MTX" ]]; then
    echo "[SKIP] not found: $MTX"
    continue
  fi

  N=$(awk '
```

```
    BEGIN{ok=0}
    /^[[:space:]]*%/{next}
    NF>=3{print $1; ok=1; exit}
    END{if(!ok) print 0}
  ' "$MTX")

  if [[ "$N" -le 0 ]]; then
    echo "[FAIL] bad header: $MTX"
    fail=$((fail+1))
    continue
  fi

  echo "===== RUN: $MTX (N=$N) ====="
  if "$BIN" --market "$MTX" --src all --num_runs "$N"; then
    echo "[OK] $MTX"
    ok=$((ok+1))
  else
    echo "[FAIL] $MTX"
    fail=$((fail+1))
  fi
done

echo "done: ok=$ok fail=$fail total=${#MTX_FILES[@]}"
```

# 4 Gunrock 与 EGGPU 复现实验结果

本章汇总我们基于上述运行脚本在多个数据集上复现得到的 Gunrock 与 EGGPU 性能结果，用于比较两者在全源精确 $BC$ 任务上的端到端耗时，并作为后续 GPU 加速实现的基线参考。实验在单张 NVIDIA Tesla V100-SXM2-32GB 服务器上进行；在无权图设置下，我们对每个数据集计算全源精确 BC，并记录整体运行时间（毫秒）。本次复现实验涉及的数据集包括 `wiki-Vote`、`wiki-Talk`、`web-Google`、`web-BerkStan` 与 `cit-Patents`。各数据集的规模信息见表 2，对应的运行耗时见表 3（其中 $n = |V|$ 为顶点数，$m = |E|$ 为边数）。

表 3 给出了 Gunrock 与 EGGPU 在五个数据集上的全源精确 BC 端到端耗时结果。总体上，两者的相对优势具有明显的数据集依赖性：在规模最小的 `wiki-Vote`（$n = 7{,}115$, $m = 103{,}689$）上，EGGPU 仅需 343 ms，而 Gunrock 为 7,048 ms（约 20.5× 更快），说明在小规模图上整体耗时更可能受框架启动、调度与内存管理等固定开销影响。对于边规模在 5–8M 的三组数据（`wiki-Talk`、`web-Google` 与 `web-BerkStan`），Gunrock 整体占优：分别达到 1.56×、2.11× 与 4.52× 的加速比（相对 EGGPU）。而在边数最大的 `cit-Patents`（$m = 16{,}518{,}948$）上，EGGPU 用时 2,561,153 ms，略快于

表 2: 数据集基本信息

| 数据集 | $n = \|V\|$ | $m = \|E\|$ |
|---|---|---|
| wiki-Vote | 7,115 | 103,689 |
| wiki-Talk | 2,394,385 | 5,021,410 |
| web-Google | 875,713 | 5,105,039 |
| web-BerkStan | 685,230 | 7,600,595 |
| cit-Patents | 3,774,768 | 16,518,948 |

表 3: Gunrock 与 EGGPU 在不同数据集上的全源精确 BC 运行耗时（ms）

| 数据集 | EGGPU（ms） | Gunrock（ms） |
|---|---|---|
| wiki-Vote | 343 | 7,048 |
| wiki-Talk | 1,368,194 | 876,698 |
| web-Google | 1,877,924 | 889,555 |
| web-BerkStan | 7,643,941 | 1,691,990 |
| cit-Patents | 2,561,153 | 3,065,440 |

Gunrock 的 3,065,440 ms（约 1.20×）。上述现象表明，两套实现的性能瓶颈与优势并非单调随规模变化，可能同时受到图结构特征、算子实现细节以及端到端开销分解的影响。因此，在后续 GPU 加速方案评估中，我们同时保留 Gunrock 与 EGGPU 作为基线对照，以更全面地刻画不同规模与结构图上的加速收益。

# 5 未来工作计划

在现有复现结果基础上，下一步工作将沿两条主线推进。第一条主线是将 Gunrock 在工程层面对多源/全源 BC 的处理方式移植到 Tigr：在不改变其核心计算流程的前提下，补齐源点集合的统一管理与全源调度机制，使其能够在一次运行中覆盖 $0, 1, \ldots, |V|-1$ 的全部源点，从而形成可直接用于全源精确 BC 的实现。第二条主线面向 Galliot：由于其未公开源码而难以直接复现，我们将以论文中给出的技术细节为依据，逐步还原其关键实现与实验设置，构建可复用的基线实现，并在相同硬件与数据集条件下与现有 Gunrock/EGGPU 的复现结果进行对比评估。