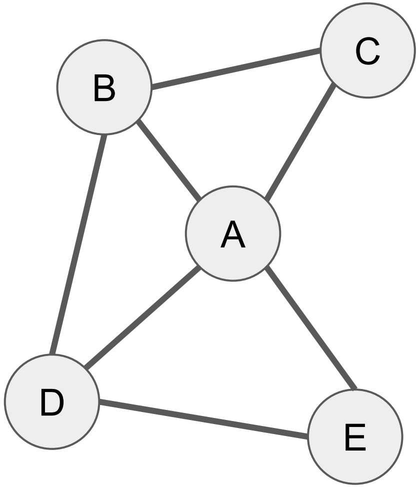
Triangle Listing

Team 02

The Problem: List All Triangles in an Undirected Graph

- Undirected simple graph
- A triangle: three nodes with pairwise edges
  - {A, B, C}, {A, B, D}, {A, D, E}
- Listing vs counting
- Useful for real-world network analysis



Schank, T., Wagner, D. (2005). Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study.

Overview

Algorithm Implementations

Instrumentation & Benchmarking

Graph Generation and “Adjusting” Real World Graphs

Optimization Ideas

Edge Iterator: Finds Shared Neighbors of Edges

- For every edge {u, w}
- If node v is in both Adj(u) and Adj(w)
- Then {u, v, w} is a triangle

- We store the graph as adjacency lists
- Triangle test: Adj(u) ∩ Adj(w)
  - Sorted Adj(u) and Adj(w) can be intersected in linear time
  - We use quicksort to preprocess them
  - Included in the total execution time

- Only consider edges (u,w) s.t. u < w, to avoid counting duplicate triangles
- Equivalent to transforming the undirected graph to a directed one

Schank, T., Wagner, D. (2005). Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study.

Forward: Dynamically Grow Neighbor Lists

- Explicitly store in-edges of w in a dynamic structure A(w)
  - After (u,w) is visited, add u to A(w)
  - |A(w)| <= d_in(w) <= |Adj(w)| = d(w)
  - A(w) uses the same data structure as Adj(w)
  - Adj(w) is sorted => A(w) is also sorted
- Triangle test: A(u) ∩ A(w) instead of Adj(u) ∩ Adj(w)

Algorithm 1: *forward*

Input: ordered list of vertices $(1, \dots, n)$, Adjacencies $Adj(v)$
Data: Node Data: $A(v)$;
for $v \in V$ **do**
  $A(v) \leftarrow \emptyset$
for $s \in (1, \dots, n)$ **do**
  for $t \in Adj(s)$ **do**
    if $s < t$ **then**
      foreach $v \in A(s) \cap A(t)$ **do**
        output triangle $\{v, s, t\}$;
      $A(t) \longleftarrow A(t) \cup \{s\}$;

Schank, T., Wagner, D. (2005). Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study.

Forward Hashed: Test Neighbors with Hash Tables

Forward Hashed

- Use a hash container for A(w)
- Triangle test: A(u) ∩ A(w)
  - Use the smaller hash table to probe the larger one
  - If hash table lookups take O(1) time, the intersection takes O(min{d_in(u), d_in(w)})

Schank, T., Wagner, D. (2005). Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study.

Instrumentation & Benchmarking

- Templated the Index type used in the implementations to count Integer operations
- Benchmarking framework:
  - Loads the graph and create helper data structures.
  - Run Instrumented version
  - Reload the graph
  - Do `$WARMUP` warmup runs
  - Run `$PHASES` phases with each `$RUNS` iterations and time the cycles

```
./benchmark -num_warmups $WARMUP -num_runs $RUNS -num_phases $PHASES -o $OUTPUTDIR/$graph.csv
-algorithm edge_iterator,forward,forward_hashed -graph $INPUTDIR/$graph.txt
```

Benchmarking - Hardware

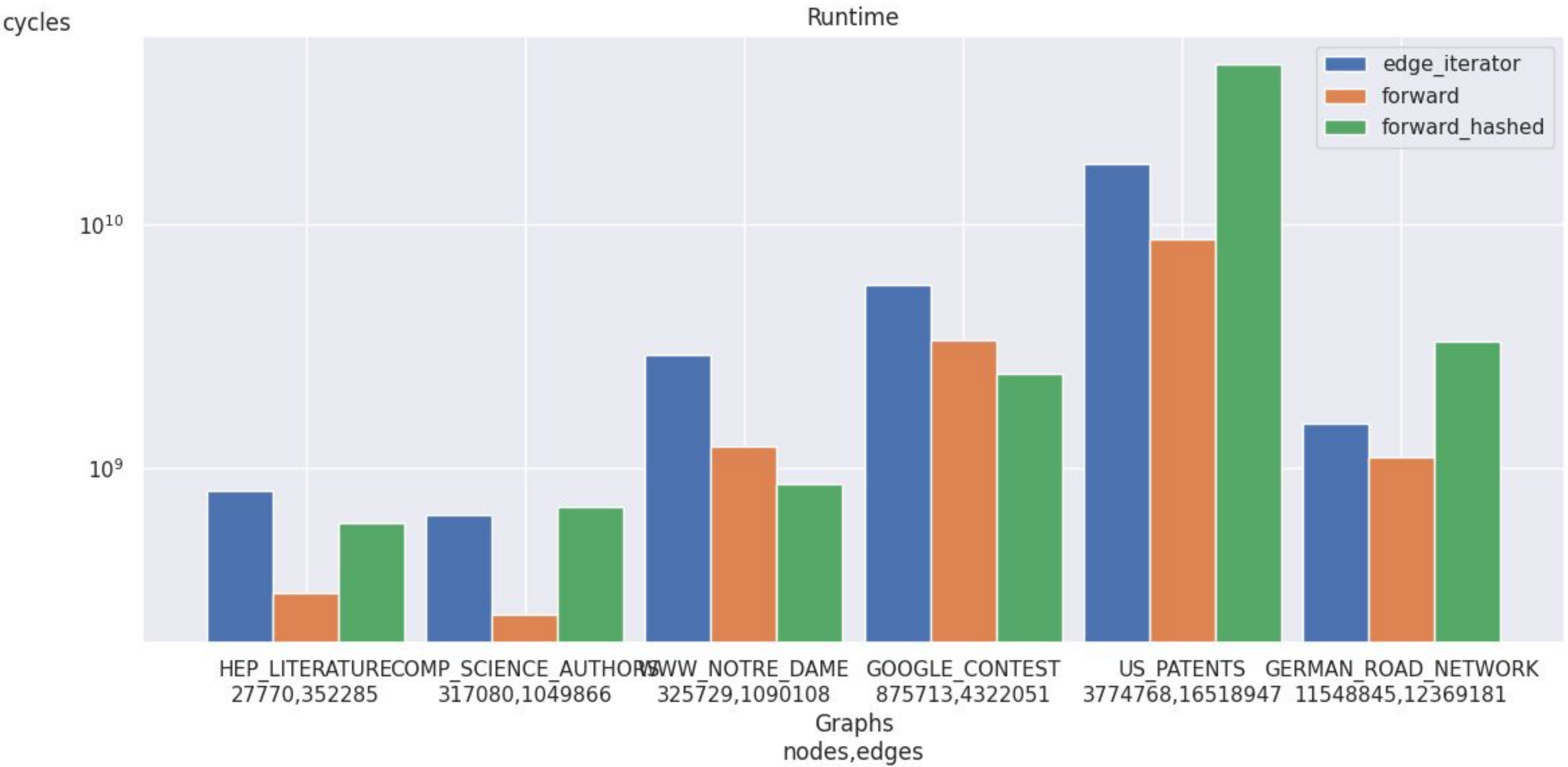
- Intel Core i9-9900K - Coffe Lake
- 4 ALU-Ports: Peak Performance 4 Integer Arithmetic ops per cycle (4x4 Vec)
- Cache Size:
  - L1: 32 KiB, 8-way set associative per core
  - L2: 32 KiB, 8-way set associative per core
  - L3: 16 Megabyte, 16-way set associative shared

Real-World Graphs

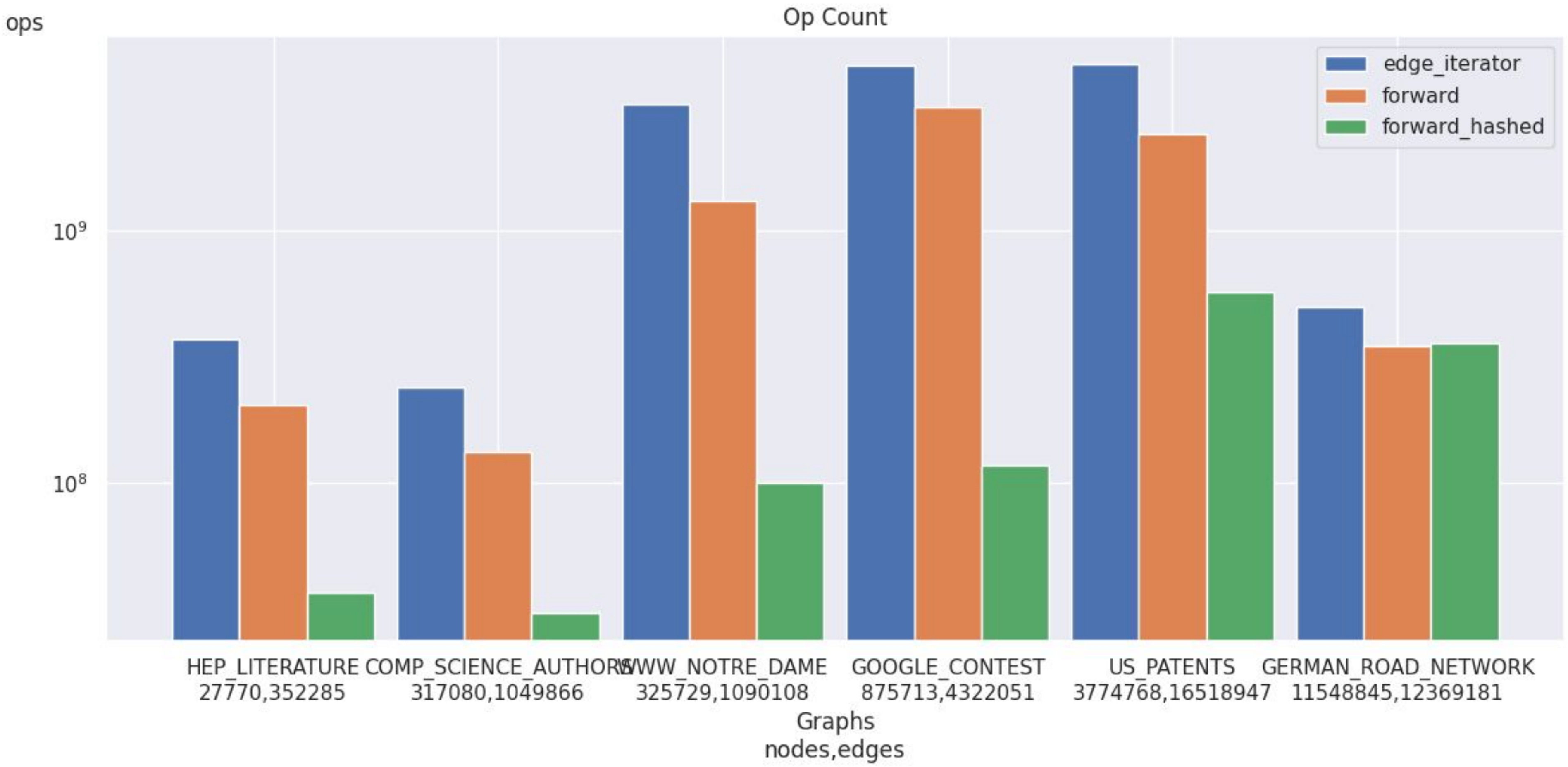
- German Road Network (2x larger)
- Computer Science Co-Authorship (roughly equal)
- Google Contest ((2x, 9x) larger)
- HEP (High Energy Particle Physics) Citation graph (roughly equal)
- Web page links nd.edu domain (equal)
- US Patent Citations (equal)

Graph Transformation and Filtering

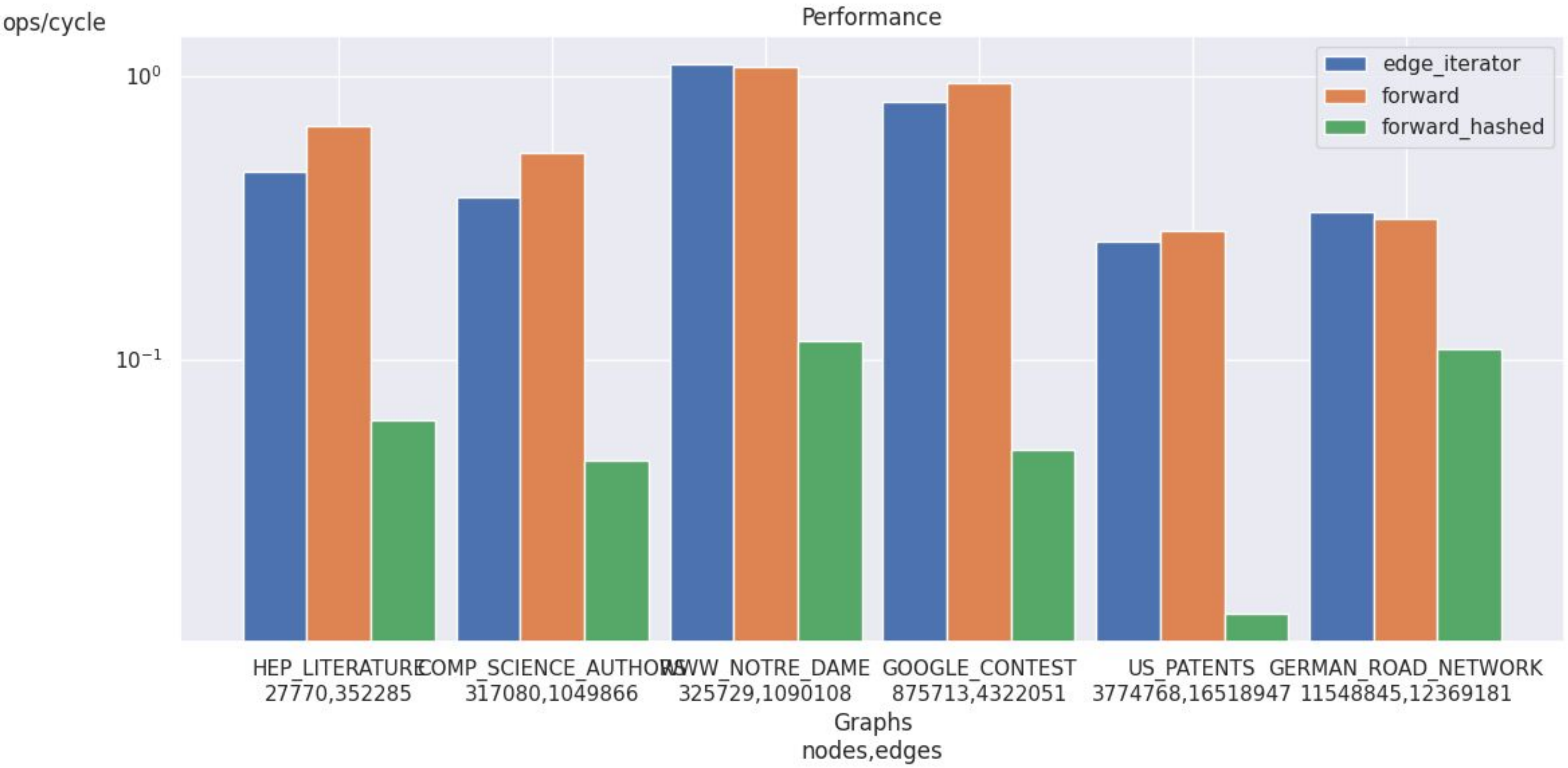
- Transformations
  - Output a list of edges for each node
  - Directed -> Undirected
    - Treat edges as undirected and add missing endpoints to respective adjacency lists
- Filtering
  - Remove Multi - Edges
  - Remove Loops



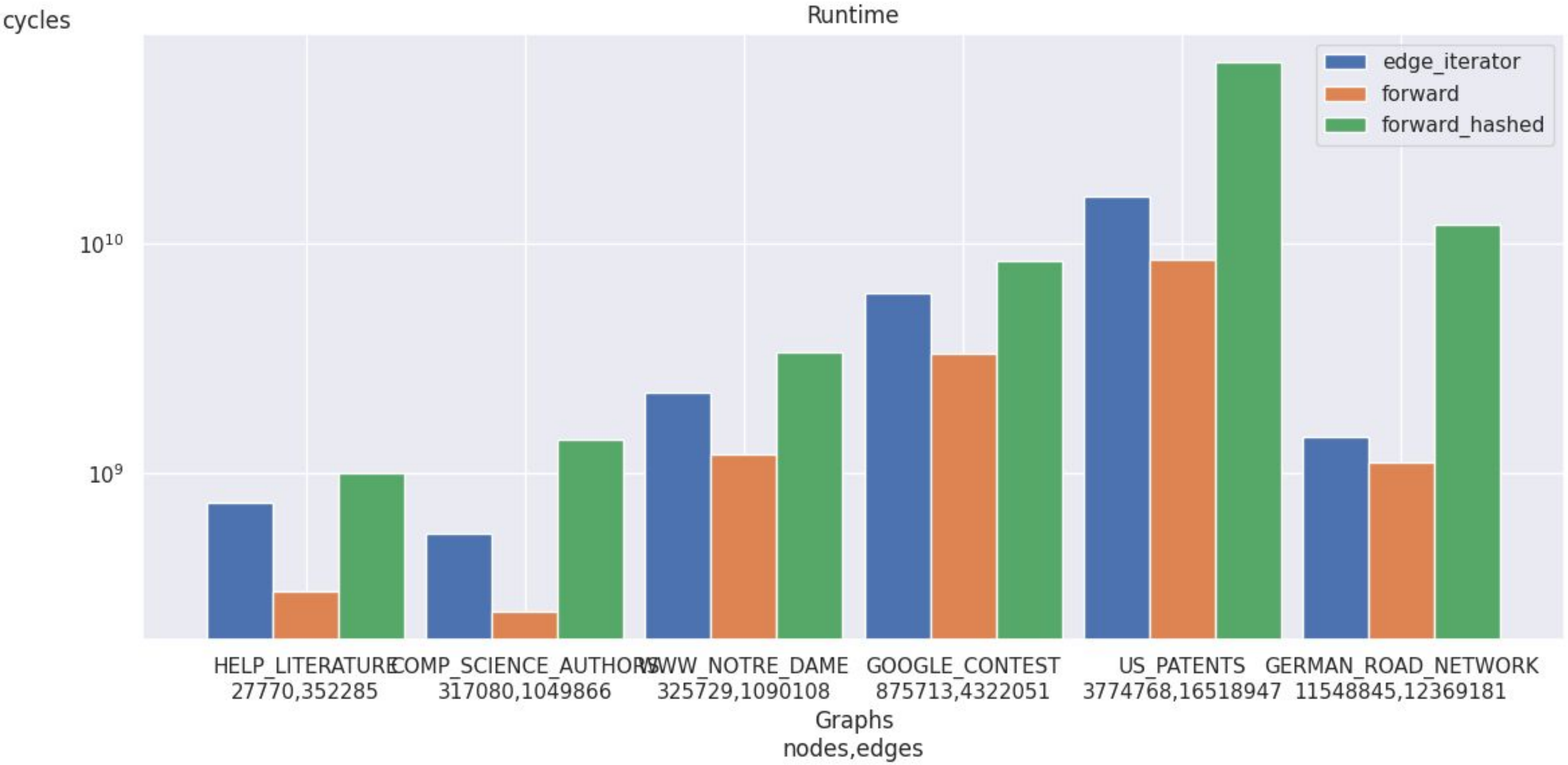
Runtime
cycles
10^{10}
10^{9}
edge_iterator
forward
forward_hashed
HEP_LITERATURE
27770,352285
COMP_SCIENCE_AUTHORS
317080,1049866
WWW_NOTRE_DAME
325729,1090108
GOOGLE_CONTEST
875713,4322051
US_PATENTS
3774768,16518947
GERMAN_ROAD_NETWORK
11548845,12369181
Graphs
nodes,edges

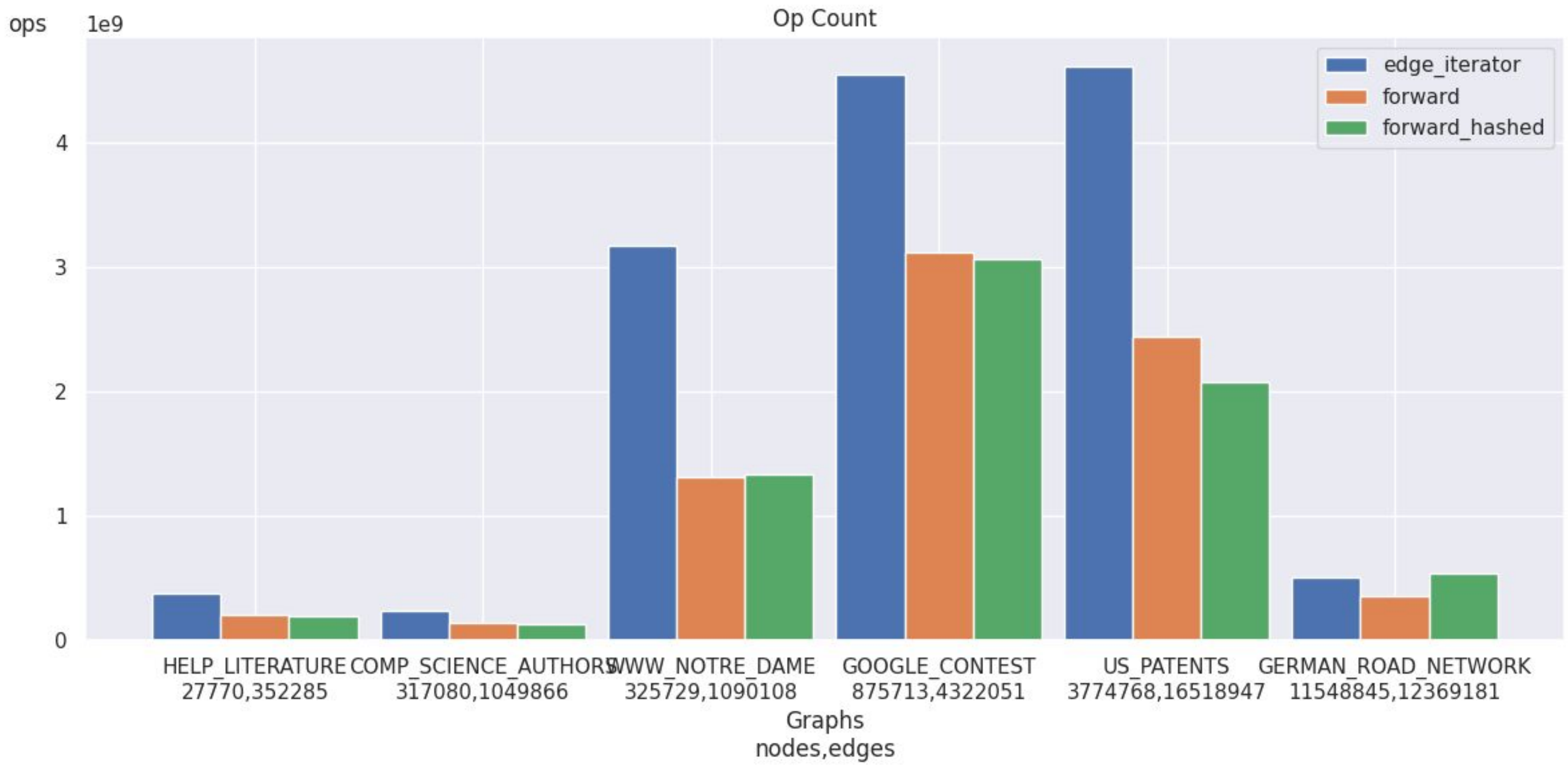


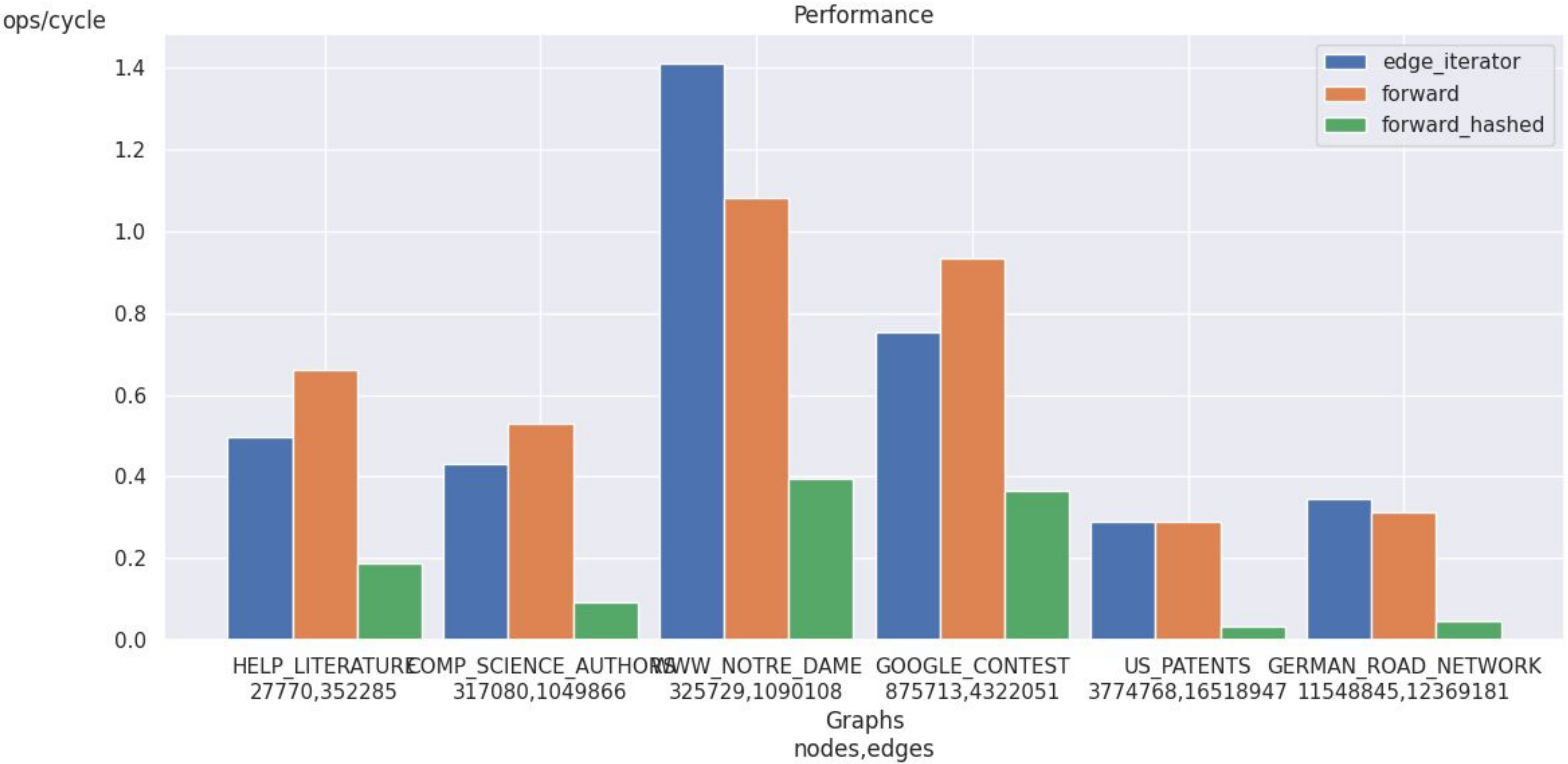
Op Count
ops
10^9
10^8
edge_iterator
forward
forward_hashed
HEP_LITERATURE
27770,352285
COMP_SCIENCE_AUTHORS
317080,1049866
WWW_NOTRE_DAME
325729,1090108
GOOGLE_CONTEST
875713,4322051
US_PATENTS
3774768,16518947
GERMAN_ROAD_NETWORK
11548845,12369181
Graphs
nodes,edges



Performance
ops/cycle
10^0
10^{-1}
edge_iterator
forward
forward_hashed
HEP_LITERATURE
27770,352285
COMP_SCIENCE_AUTHORS
317080,1049866
WWW_NOTRE_DAME
325729,1090108
GOOGLE_CONTEST
875713,4322051
US_PATENTS
3774768,16518947
GERMAN_ROAD_NETWORK
11548845,12369181
Graphs
nodes,edges


Runtime
cycles
edge_iterator
forward
forward_hashed
10^{10}
10^{9}
HELP_LITERATURE
27770,352285
COMP_SCIENCE_AUTHORS
317080,1049866
WWW_NOTRE_DAME
325729,1090108
GOOGLE_CONTEST
875713,4322051
US_PATENTS
3774768,16518947
GERMAN_ROAD_NETWORK
11548845,12369181
Graphs
nodes,edges

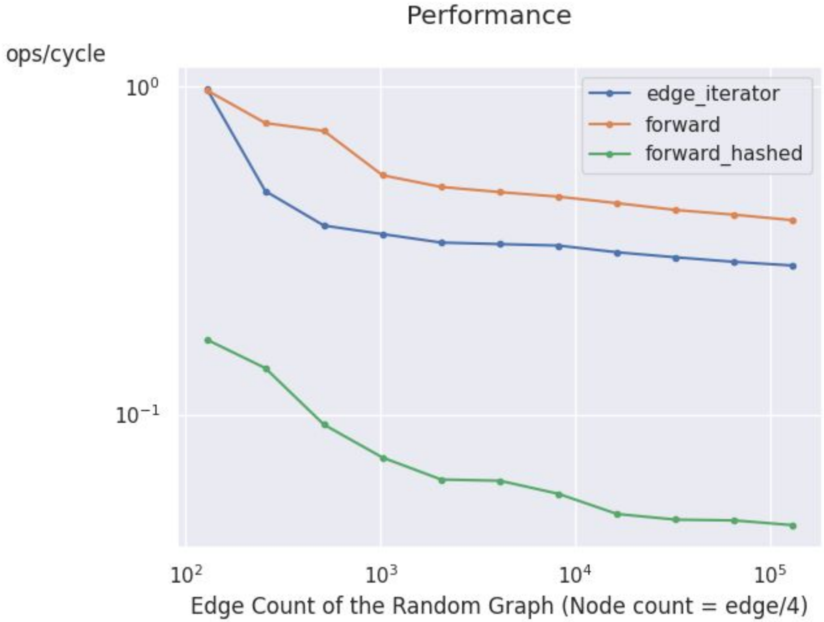

Op Count
ops
1e9
0
1
2
3
4
edge_iterator
forward
forward_hashed
HELP_LITERATURE
27770,352285
COMP_SCIENCE_AUTHORS
317080,1049866
WWW_NOTRE_DAME
325729,1090108
GOOGLE_CONTEST
875713,4322051
US_PATENTS
3774768,16518947
GERMAN_ROAD_NETWORK
11548845,12369181
Graphs
nodes,edges

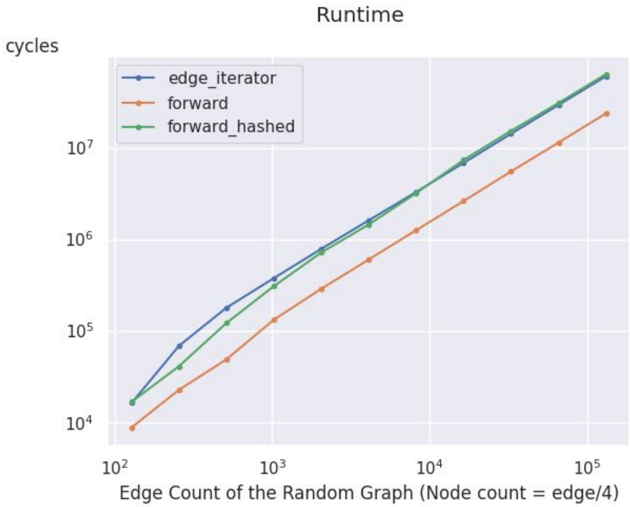


Performance
ops/cycle
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
edge_iterator
forward
forward_hashed
HELP_LITERATURE
27770,352285
COMP_SCIENCE_AUTHORS
317080,1049866
WWW_NOTRE_DAME
325729,1090108
GOOGLE_CONTEST
875713,4322051
US_PATENTS
3774768,16518947
GERMAN_ROAD_NETWORK
11548845,12369181
Graphs
nodes,edges

Random Graph Results

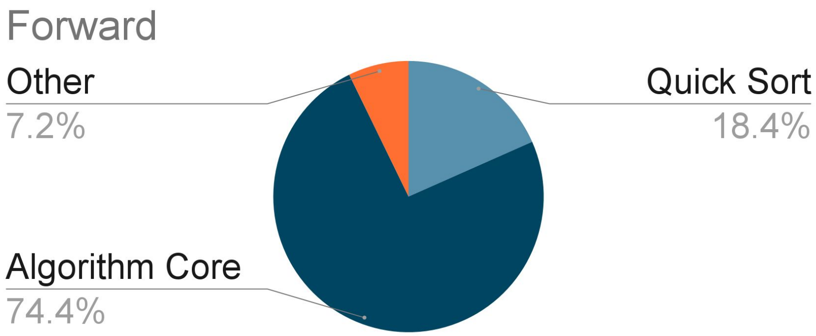
- Randomly distribute connect nodes until all edges have been created

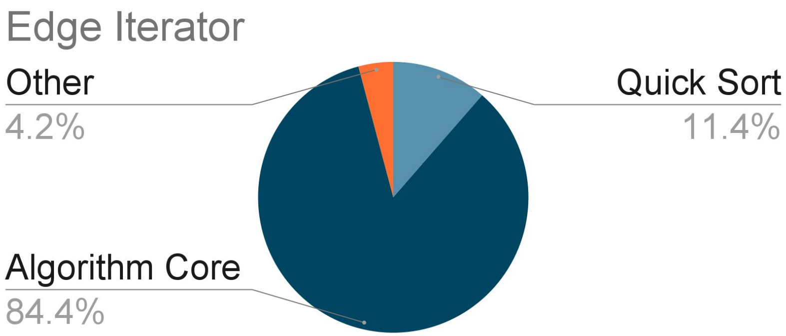


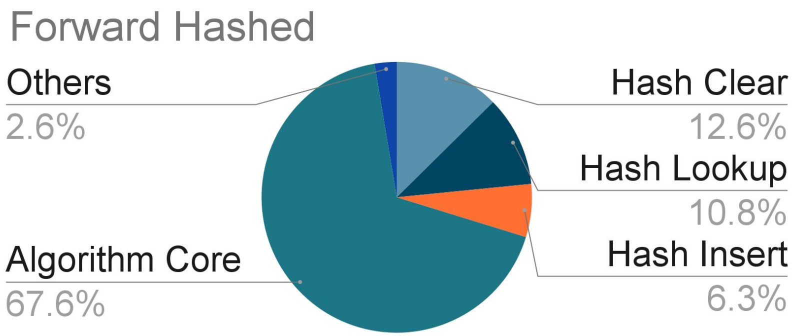


Profiling Result

- Algorithm cores take most of the time.
- There is also space for optimization in quick sort and hash table.







Optimization Ideas

- Integer size
    - 8-byte ints or smaller ints
- Indirections
    - Pointers in structs and linked lists
    - Reduce the number of indirections in the data structures
- Data Structures
    - Data structure optimizations towards different graph properties (density, max degree, etc.)
    - More compact
    - More efficient for preprocessing, triangle tests and hash table operations

Optimization Ideas

- Function Inlining
- ILP
  - 4 ports each with 1 ALU execution unit
  - Set intersections are independent, but the naive implementation of set intersection is blocking
  - Loop unrolling
- Cache
  - Temporal locality of adjacency lists: blocking
  - Update dynamic structure
- Branching
  - Branching: quicksort, undirected→directed transformation, set intersection, hash table lookup
  - Reduce branch mispredictions

Optimization Ideas

- Hash Table
  - Hash functions: modulo
  - Hash container size
  - Collision resolution: separate chaining vs. open addressing
- SIMD
  - Use SIMD instructions to accelerate set intersections and hash table operations

Questions

- What exactly should be counted as an operation for the performance plots?
- Benchmark listing vs counting?
- Warmup runs?
- Should we time resetting the helper datastructure?
- How should we un-sort the edges again, this basically doesn't allow us to have more than one run per phase (which is probably fine for big graphs).
- Memory allocation?
- To what extent can we change the graph representation and operations so that the algorithms can still be considered as the algorithms given in the paper?
- C++ for template in the main algo body?