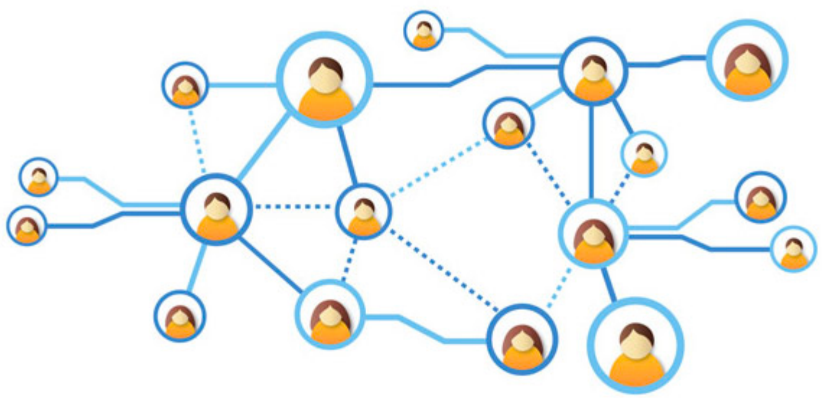
Parallel Five-Cycle Counting Algorithms

Louisa Ruixue Huang (MIT CSAIL)

Jessica Shi (MIT CSAIL)

Julian Shun (MIT CSAIL)

Graph Processing

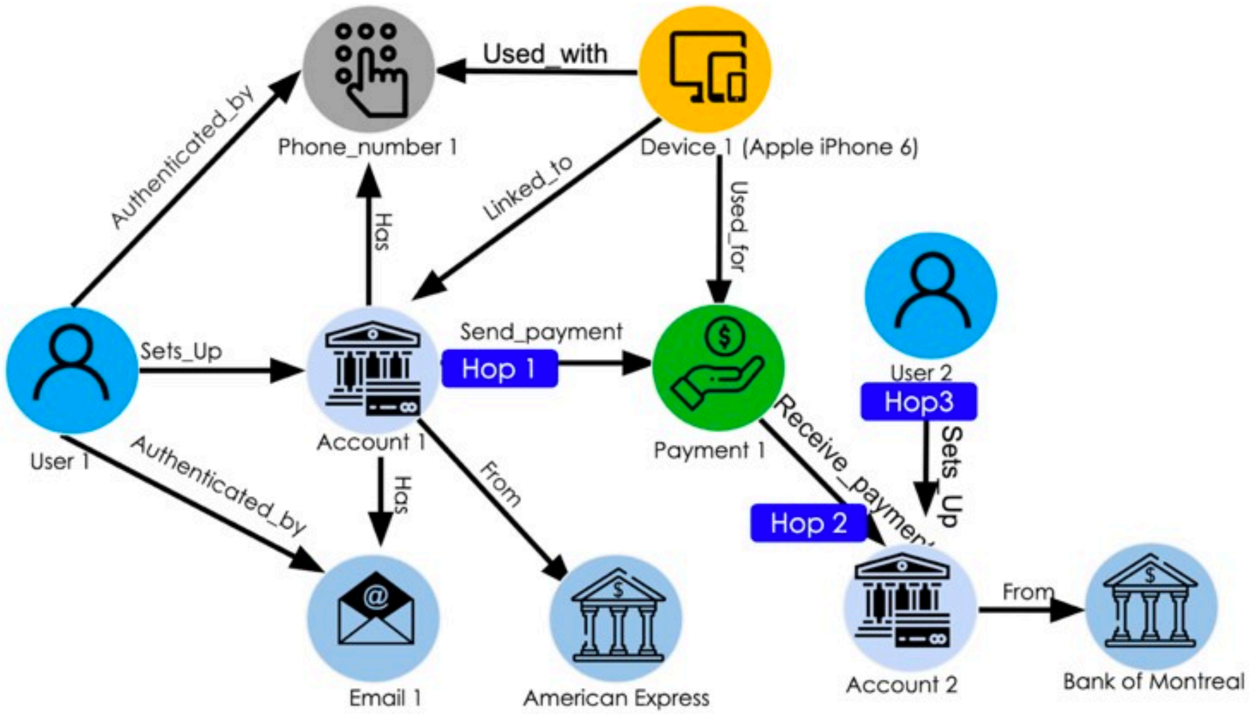


Social Network

https://blog.soton.ac.uk/skillted/2015/04/05/graph-theory-for-skillted/

Road Network

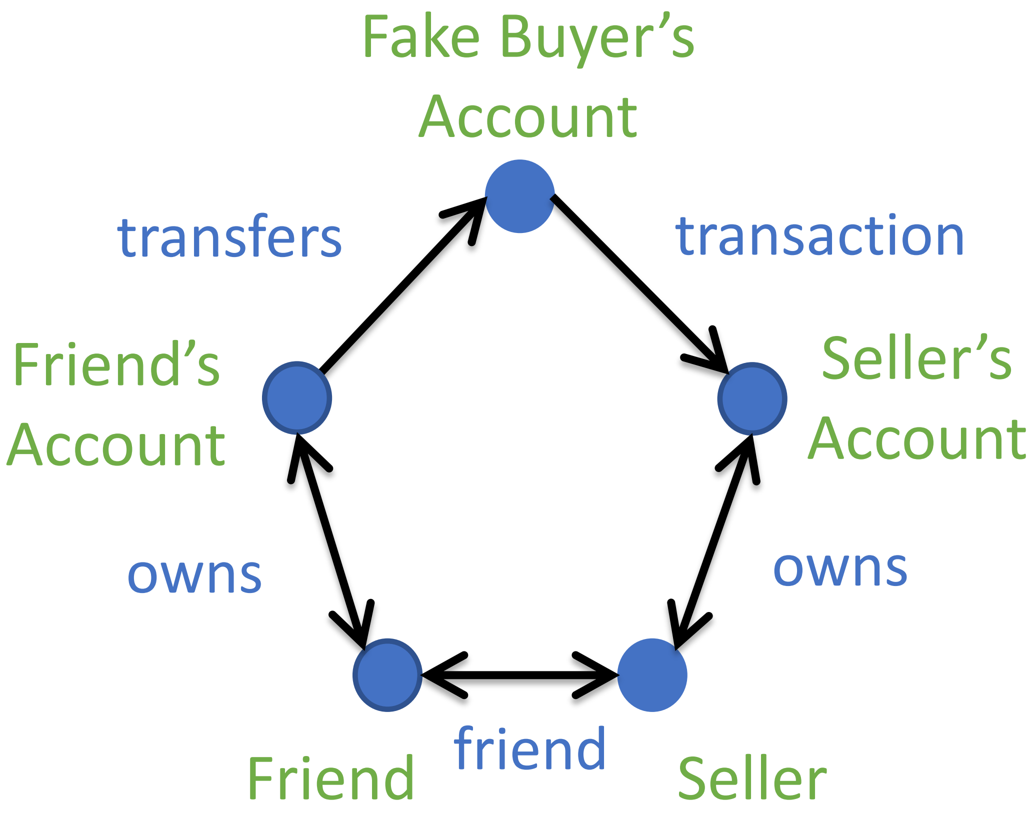
Data-driven Modeling of Transportation Systems and Traffic Data Analysis During a Major Power Outage in the Netherlands



Financial Transactions

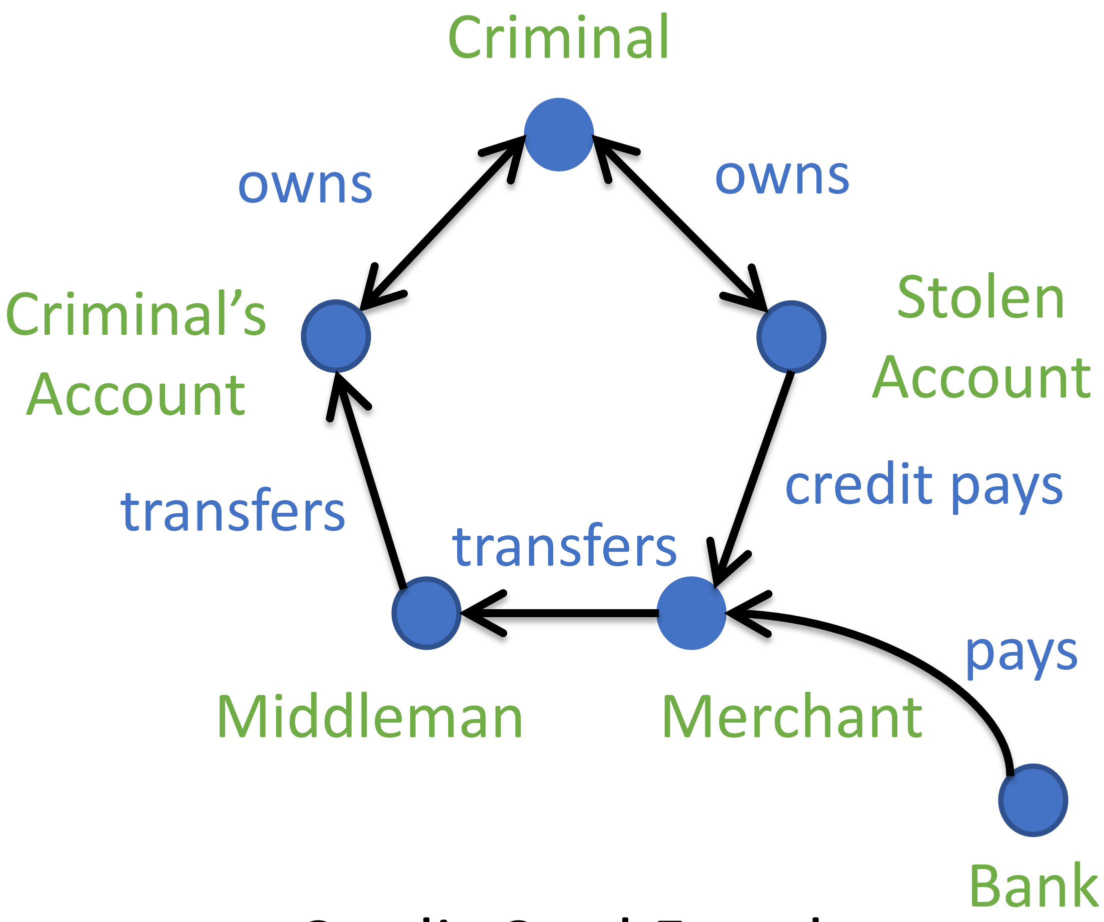
https://www.rtinsights.com/how-the-worlds-largest-banks-use-advanced-graph-analytics-to-fight-fraud/

Five-Cycle Counting



Merchant Fraud

Real-time Constrained Cycle Detection in Large Dynamic Graphs (Qiu et al., 2018)



Credit Card Fraud

Real-time Constrained Cycle Detection in Large Dynamic Graphs (Qiu et al., 2018)

Five-Cycle Counting

- k-cycle counting is computationally intensive
  - Exponential growth in number of possible subgraphs as k increases
  - ESCAPE [1] package: Counts all five-vertex subgraphs
    - 25 – 58% of time in ESCAPE is spent on five-cycles
  - Theoretical barrier for k-cycle counting for k > 5 [2]

[1] Pinar, Seshadhri, Vishal (16)
[2] Bera, Pashanasangi, Seshadhri (20)

Parallelism

- Parallelism enables us to efficiently process large graphs

Apple, Microsoft, Intel, https://www.flickr.com/photos/66016217@N00/2556707493/, HP

Main Contributions

- Main Goal: Design and implement algorithms to efficiently count five-cycles in a graph

- First theoretically efficient parallel algorithms for counting five-cycles
- New practical optimizations for fast parallel performance
- Comprehensive evaluation
  - Outperforms previous fastest sequential implementations [1] by up to 818x
  - Up to 43x self-relative speedups

[1] Pinar, Seshadhri, Vishal (16)

Main Contributions

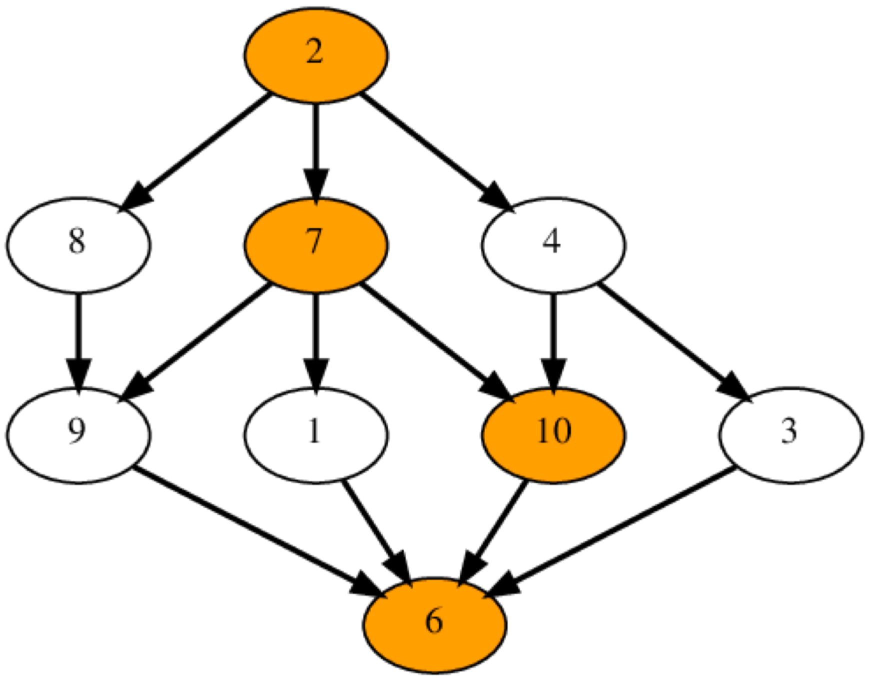
- We present two five-cycle counting algorithms that achieve the same theoretical complexity
- Based on two sequential counterparts:
  - Kowalik [1] : Theoretically efficient, based on ordered 2-paths
  - ESCAPE [2] : Based on directed 3-paths
    - (We provide an important modification to the serial ESCAPE to make it theoretically efficient)

[1] Kowalik (03)
[2] Pinar, Seshadhri, Vishal (16)

Important paradigms

- Strong theoretical bounds
  - Work = total # operations = # vertices in graph
  - Span = longest dependency path = longest directed path
  - Running Time $\leq$ (work / # processors) + O(span)
  - Work-efficient = work matches sequential time complexity

Parallel computation graph



https://www.researchgate.net/figure/Task-dependency-graph-each-node-contains-the-task-time-and-the-highlighted-tasks-form_fig1_320678407

Graph Ordering and Orientation

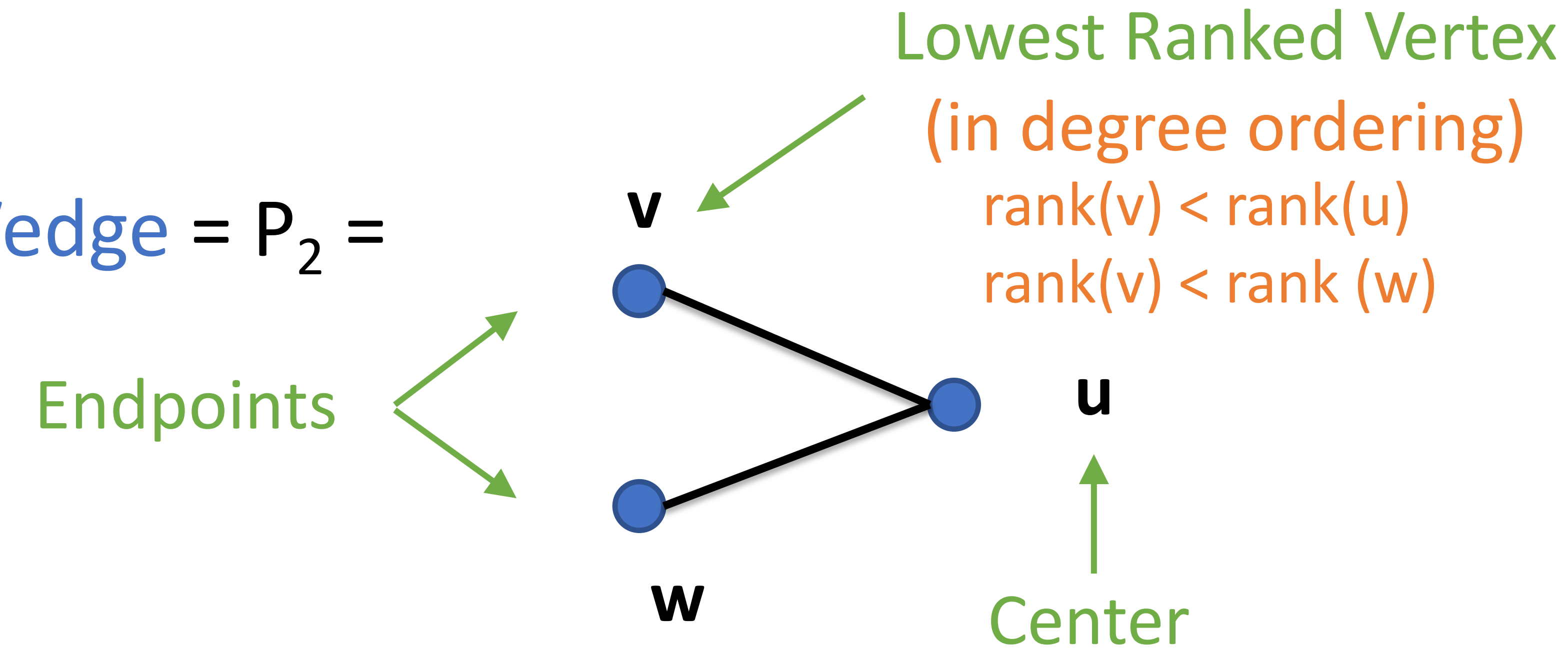
- Arboricity Orientation: Direct graph such that each vertex's out-degree is upper bounded by O(α)
  - α = arboricity/degeneracy (O($\sqrt{m}$))
  - m = # edges
  - Can compute in O(m) work, O($\log^2$ n) span [1]
- Degree Ordering: Order vertices by non-increasing degree
  - Lemma [2]: $\sum_{(u,v)\in E} \min\big(d(u), d(v)\big) \leq 2\alpha m$

[1] Shi, Dhulipala, Shun (21)
[2] Chiba, Nishizeki (85)

Parallel Five-Cycle Counting Algorithm (based on Kowalik)

Wedges

Wedge = P_2 =

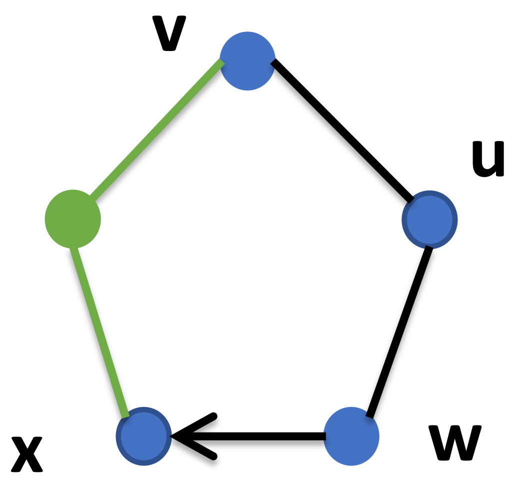


= (v, u, w)

To avoid double counting, we find all cycles from the lowest ranked vertex in the cycle

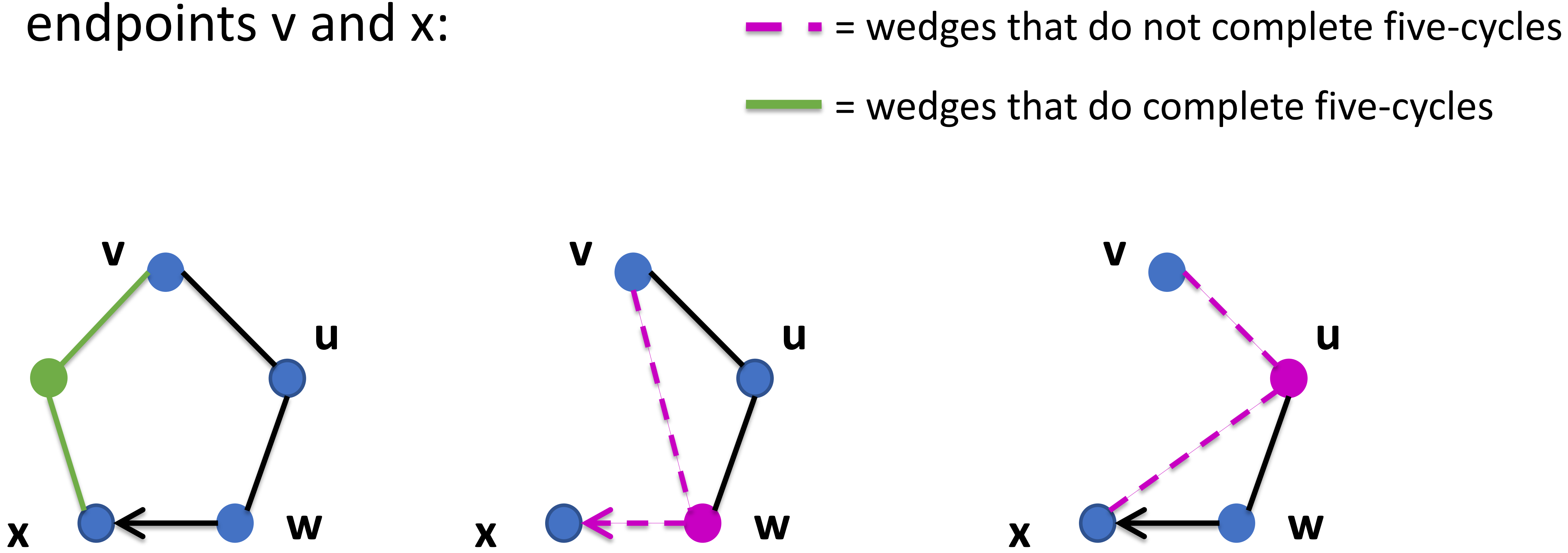
Main Idea

- Parallel for each wedge (v, u, w): (unique via degree ordering)
  - Consider now the arboricity oriented graph
  - Parallel for each arboricity directed neighbor x of w, such that x is after v in degree ordering: (unique three-path)
    - # of wedges with endpoints v and x complete the cycle



Incorrect Counting

- We must address incorrect counting when finding wedges with endpoints v and x:

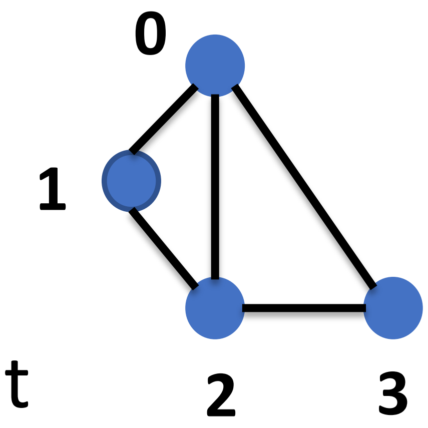


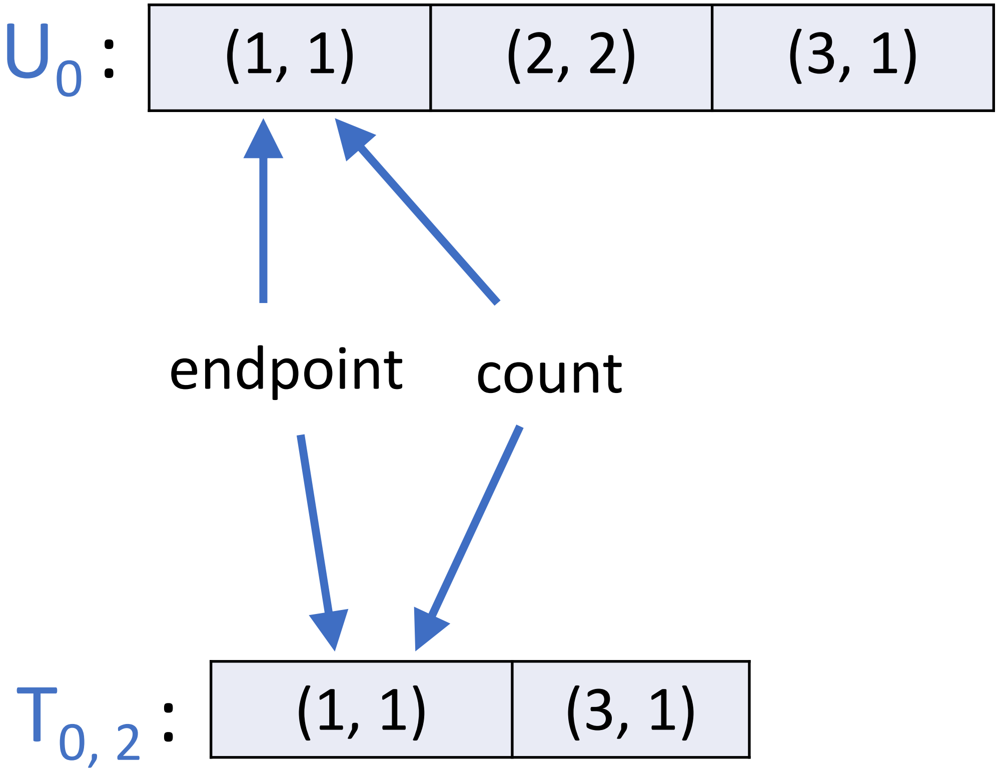
Data Structures for Wedges

- For each vertex v:
- Parallel hash table:

U_v : keys = second endpoint

values = # of wedges with endpoint v



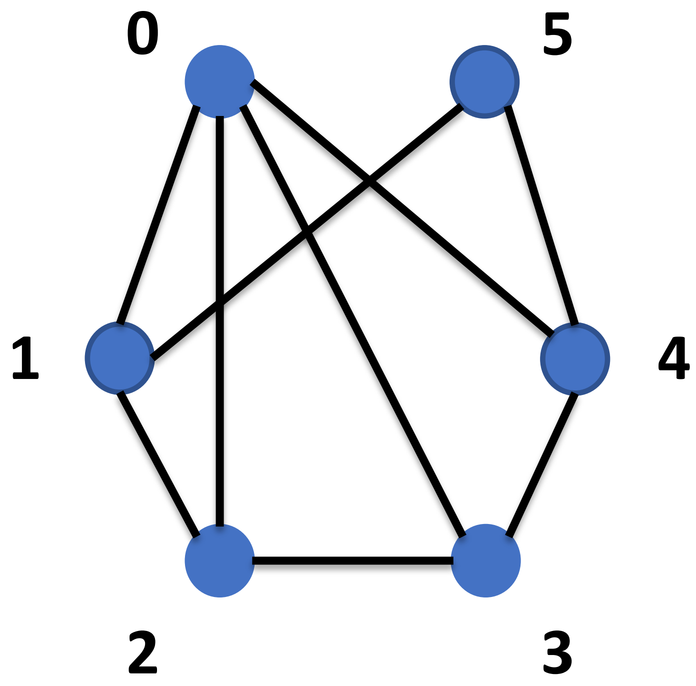


- For each pair of vertices (v, u):
- Parallel hash table:

$T_{v,\,u}$: keys = second endpoint

values = # of wedges with endpoint v and center u

Data Structures for Wedges



Vertex IDs in degree ordering

U_0 : # of wedges with endpoints (0, key)

(1, 1)	(2, 2)	(3, 2)	(4, 1)	(5, 2)

$T_{0,1}$

(2, 1)	(5, 1)

$T_{0,3}$

(2, 1)	(4, 1)

$T_{0,2}$

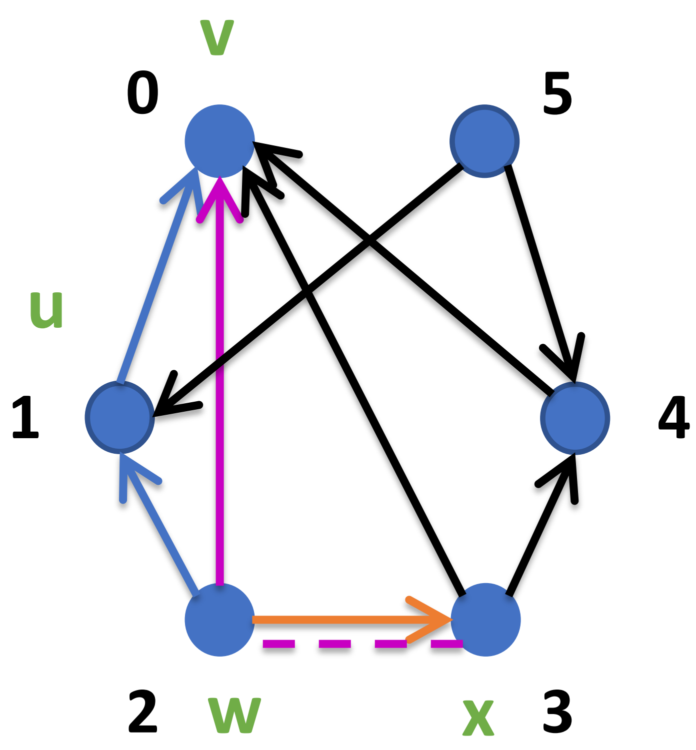
(1, 1)	(3, 1)

$T_{0,4}$

(3, 1)	(5, 1)

$T_{0,u}$: # of wedges (0, u, key)

Five-Cycle Counting Example



Vertex IDs in degree ordering,
Arrows in arboricity orientation

Wedge (v, u, w) : (0, 1, 2)

Directed edge (w, x) : (2, 3)

Number of (v, x) wedges : $U_0[3] = 2$

(v, w) is an edge : Subtract 1

$T_{v,\,u}[x] = 0$: Subtract 0

1 cycle

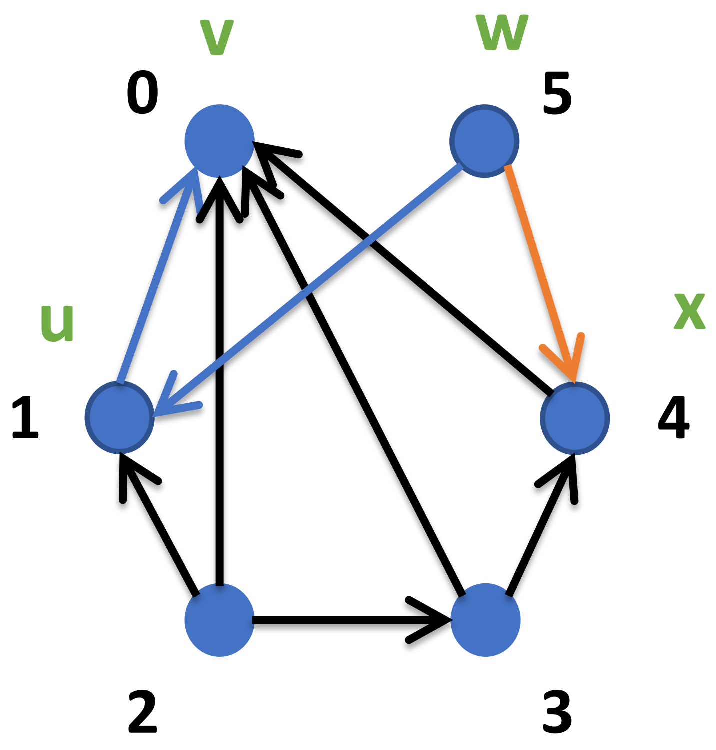
U_0 : # of wedges with endpoints (0, key)

(1, 1)	(2, 2)	(3, 2)	(4, 1)	(5, 2)

$T_{0,\,1}$: (0, 1, key)

(2, 1)	(5, 1)

Five-Cycle Counting Example



Wedge (v, u, w) : (0, 1, 5)

Directed edge (w, x) : (5, 4)

Number of (v, x) wedges : $U_0[4] = 1$

(v, w) is not an edge : Subtract 0

$T_{v,\ u}\,[x] = 0$: Subtract 0

1 cycle

U_0 : # of wedges with endpoints (0, key)

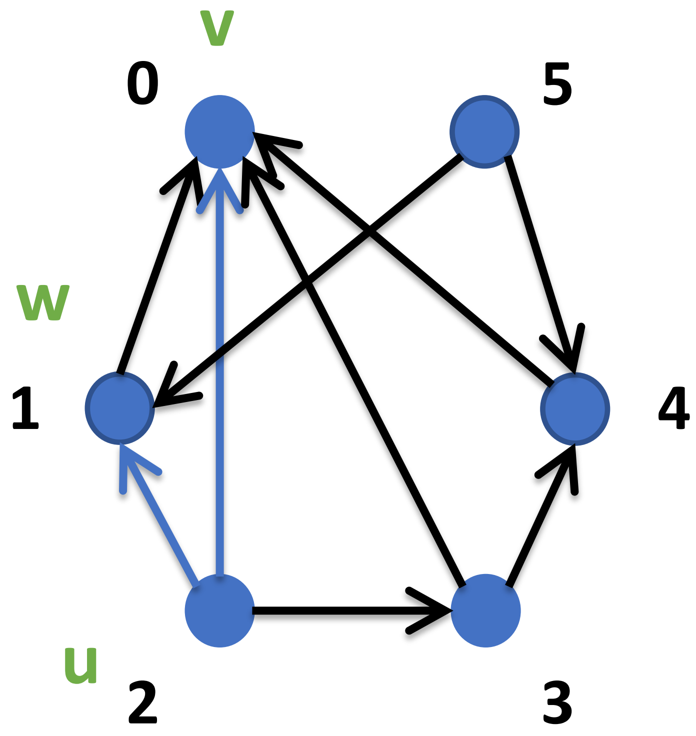
(1, 1)	(2, 2)	(3, 2)	(4, 1)	(5, 2)

$T_{0,\ 1}$: (0, 1, key)

(2, 1)	(5, 1)

Vertex IDs in degree ordering,
Arrows in arboricity orientation

Five-Cycle Counting Example



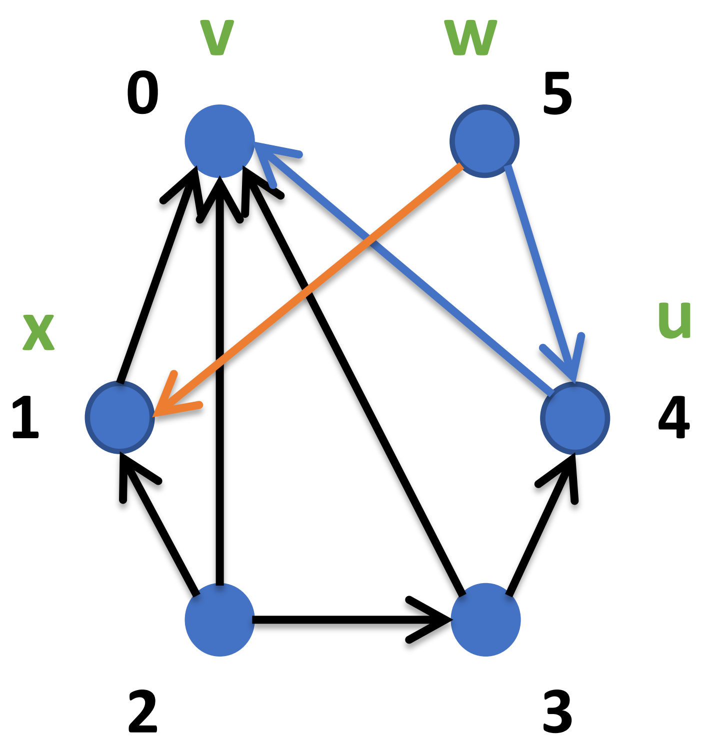
Wedge (v, u, w) : (0, 2, 1)

No directed edges (w, x)

0 cycles

Vertex IDs in degree ordering,
Arrows in arboricity orientation

Five-Cycle Counting Example



Wedge (v, u, w) : (0, 4, 5)

Directed edge (w, x) : (5, 1)

Number of (v, x) wedges : $U_0[1] = 1$

(v, w) is not an edge : Subtract 0

$T_{v,\,u}[x] = 0$: Subtract 0

1 cycle

U_0 : # of wedges with endpoints (0, key)

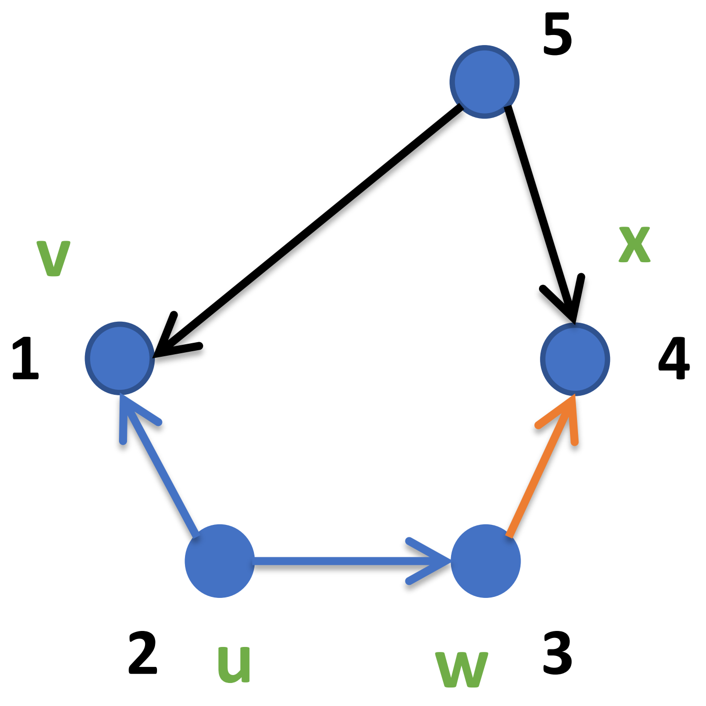
(1, 1)	(2, 2)	(3, 2)	(4, 1)	(5, 2)

$T_{0,\,4}$: (0, 4, key)

(3, 1)	(5, 1)

Vertex IDs in degree ordering,
Arrows in arboricity orientation

Five-Cycle Counting Example



Vertex IDs in degree ordering,
Arrows in arboricity orientation

Wedge (v, u, w) : (1, 2, 3)

Directed edge (w, x) : (3, 4)

Number of (v, x) wedges : $U_0[4] = 1$

(v, w) is not an edge : Subtract 0

$T_{v, u}[x] = 0$: Subtract 0

1 cycle

U_1 : # of wedges with endpoints (1, key)

(3, 1)	(4, 1)

$T_{1, 2}$: (1, 2, key)

(3, 1)

In total: 4 cycles

Theoretical Bounds

- Lemma [1] : Total # of wedges = $\sum_{(u,v)\in E} \min\big(d(u), d(v)\big) \leq 2\alpha m$
- Arboricity orientation: $O(m)$ work, $O(\log^2 n)$ span [2]
- Degree ordering: $O(n)$ work, $O(\log n)$ span whp [3]
- Constructing hash tables U, T: $O(m\alpha)$ work, $O(\log^* n)$ span whp
- Extending a wedge with a directed edge: Multiply by α for the work

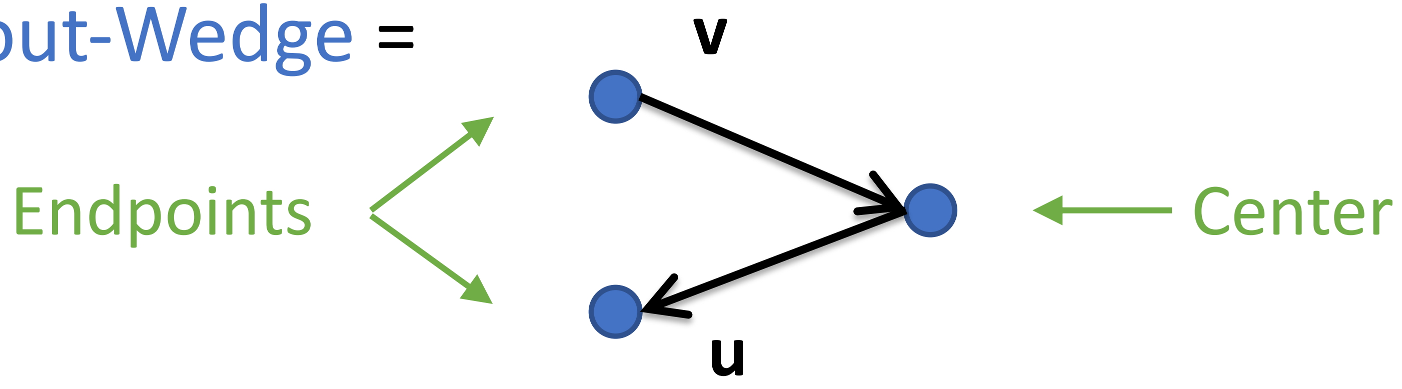
Total = $O(m\alpha^2)$ work, $O(\log^2 n)$ span whp

[1] Chiba, Nishizeki (85)
[2] Shi, Dhulipala, Shun (21)
[3] Rajasekaran, Reif (89)

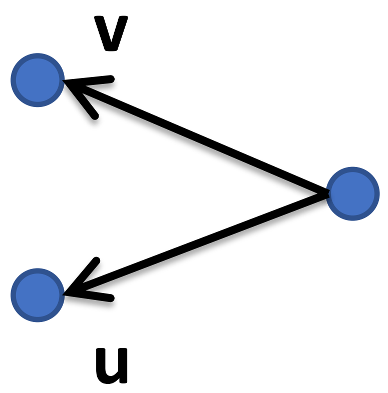
Parallel Five-Cycle Counting Algorithm (based on ESCAPE)

Arboricity Oriented Wedges

Inout-Wedge =

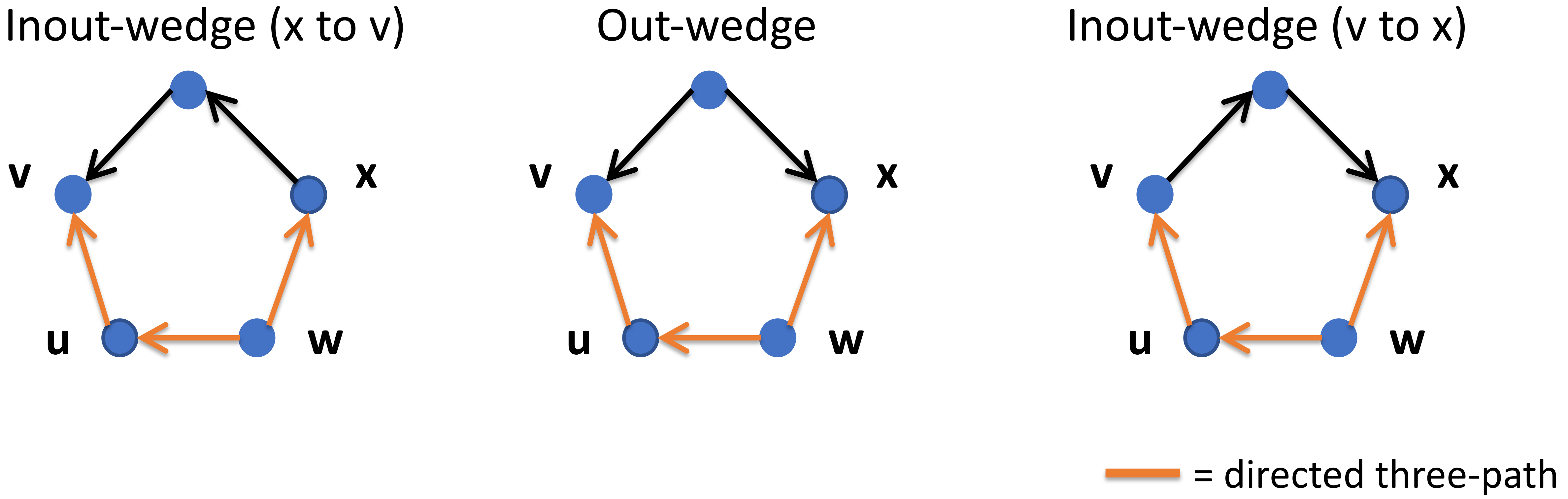


Out-Wedge =



Main Idea

- All possible acyclic orientations of directed five-cycles:

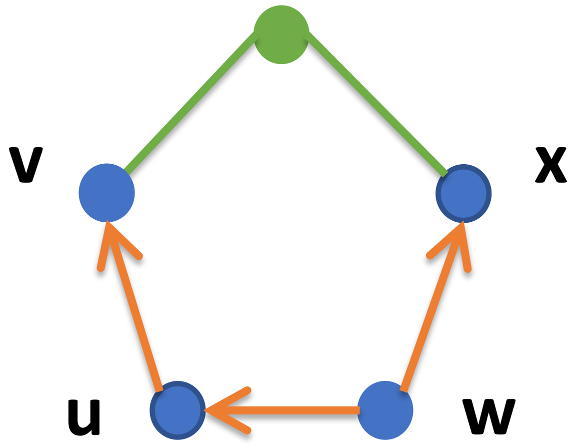


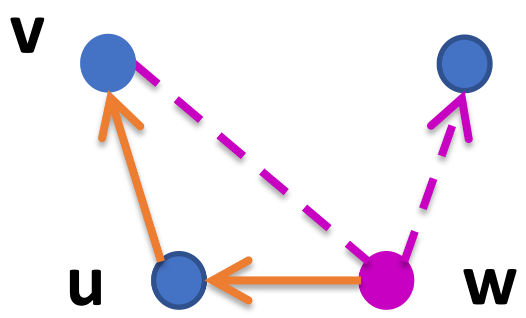
Main Idea

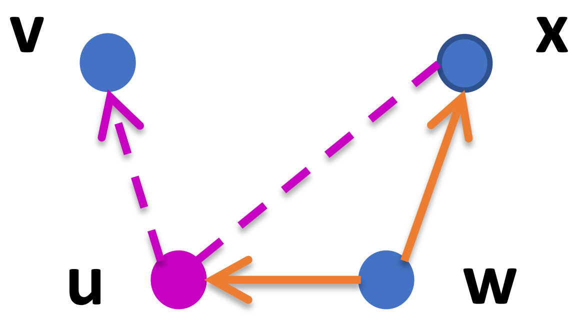
- Parallel for every ($v \leftarrow u \leftarrow w \rightarrow x$): (unique via arboricity ordering)
  - # of inout- and out-wedges with endpoints v and x complete the cycle
  - Incorrect counting (check if (w, v) or (x, u) are edges):

= wedges that do not complete five-cycles

= wedges that do complete five-cycles







Theoretical Bounds

- Arboricity orientation: $O(m)$ work, $O(\log^2 n)$ span [1]
- Constructing hash table U: $O(m\alpha)$ work, $O(\log^* n)$ span whp
- Iterating over 3-paths: $O(m\alpha^2)$ 3-paths

Total = $O(m\alpha^2)$ work, $O(\log^2 n)$ span whp

[1] Shi, Dhulipala, Shun (21)

Evaluation

Optimizations

- Thread-local Data Structures:
  - Space for parallel hash tables per vertex only allocated once per processor
- Fast Reset:
  - Additional thread-local array to mark used hash table entries
- Work Scheduling:
  - Group vertices by estimating work, such that work per group is equal
  - Estimate given by sum of degrees of neighbors
  - Parallelize between groups

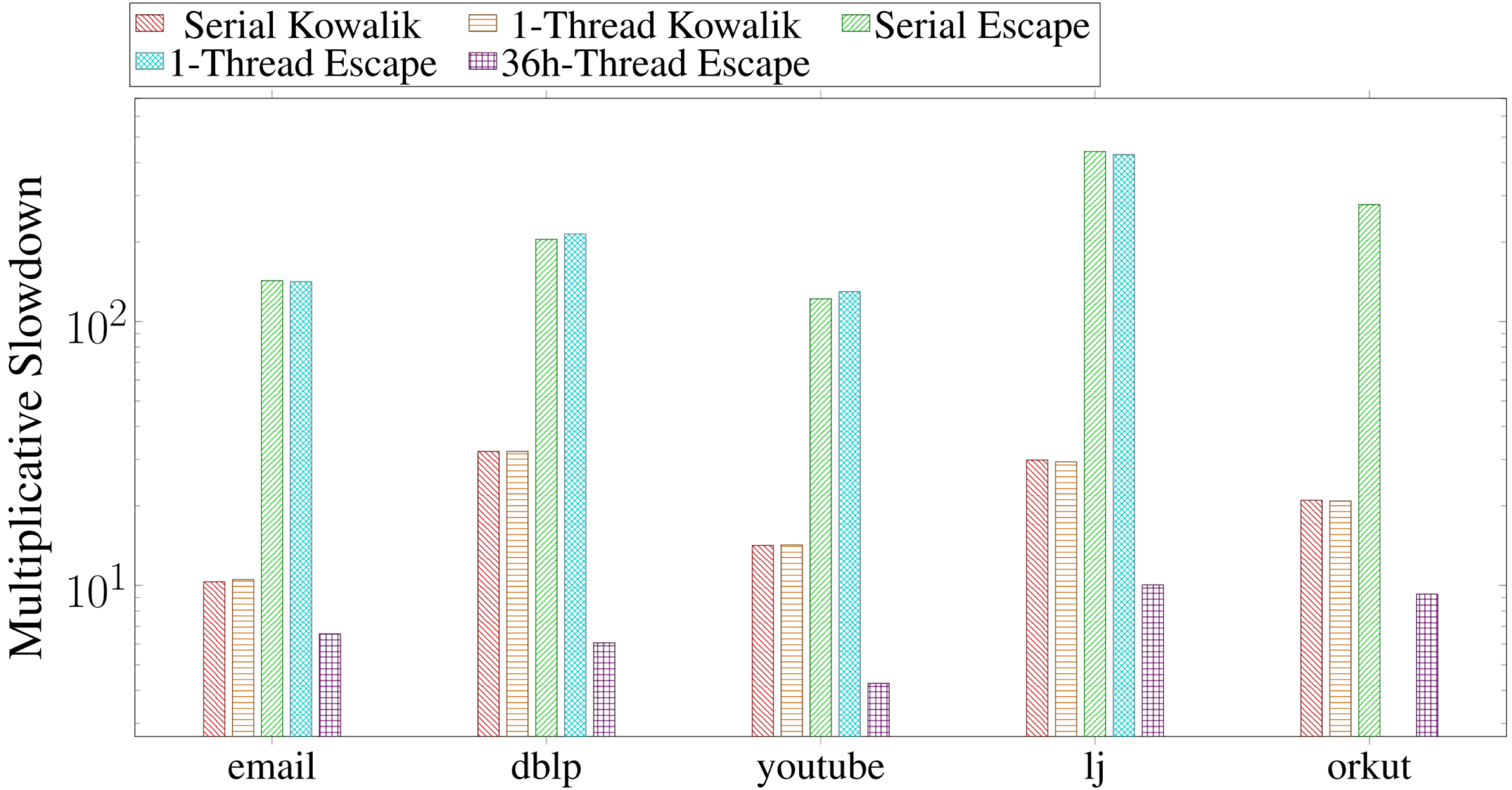
Environment

- c5.18xlarge AWS EC2 instance: dual-processor, 18 cores per processor (2-way hyper-threading), 144 GiB main memory
- Cilk Plus[1] work-stealing scheduler
- Real-world Stanford Network Analysis Platform (SNAP) graphs

Graph	# Vertices	# Edges	# 5-cycles
email	1005	32128	2.45×10^{8}
dblp	425957	2.10×10^{6}	3.44×10^{9}
youtube	1.16×10^{6}	5.98×10^{6}	3.46×10^{10}
lj	4.03×10^{6}	6.94×10^{7}	6.67×10^{12}
orkut	3.27×10^{6}	2.34×10^{8}	4.25×10^{13}
friendster	1.25×10^{8}	3.61×10^{9}	9.63×10^{13}

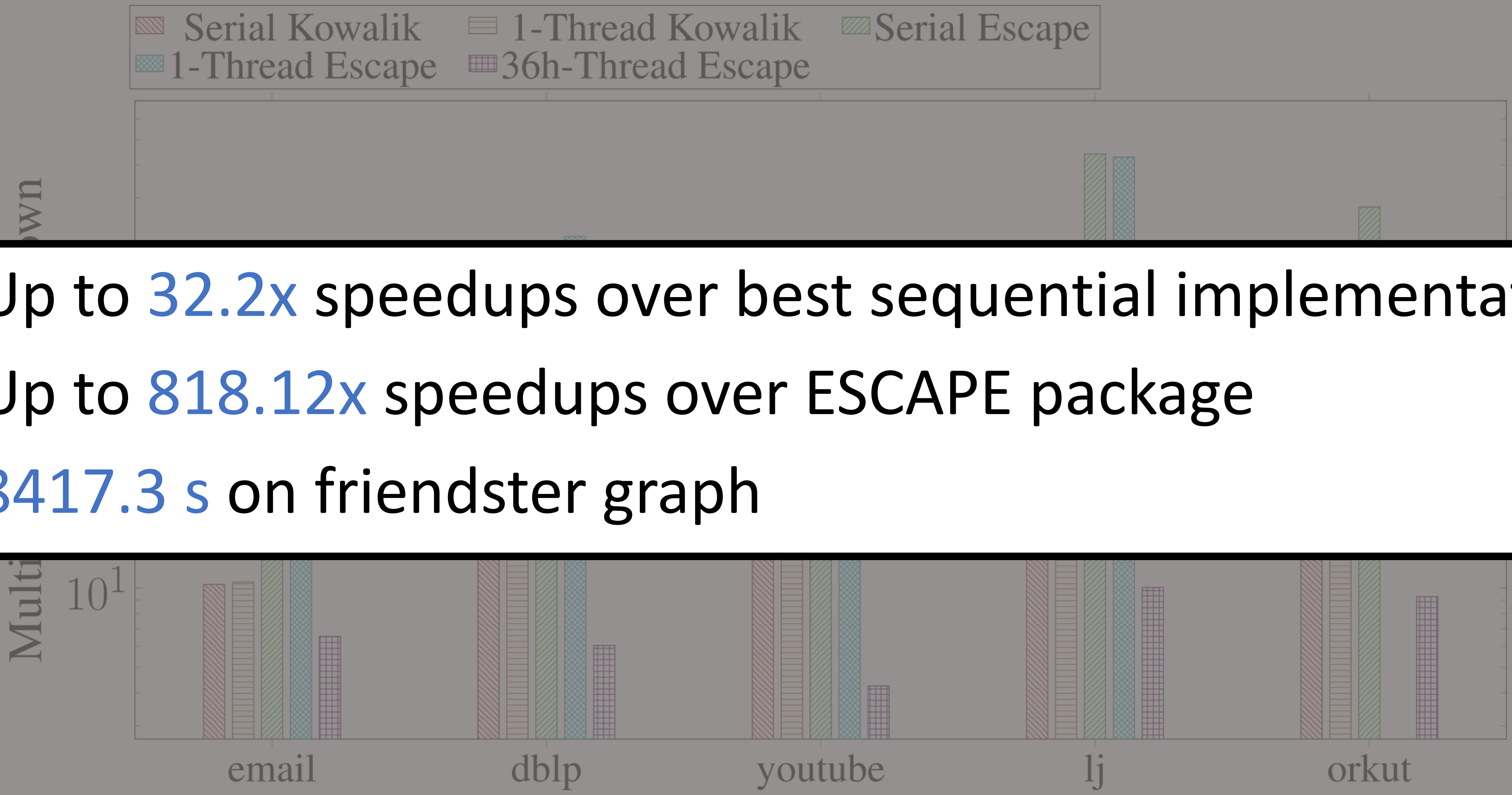
[1] Leiserson (10)

Main Running Times



36h-Thread Kowalik: (0.0025 s) (0.014 s) (0.33 s) (5.85 s) (136.98 s)
(fastest)

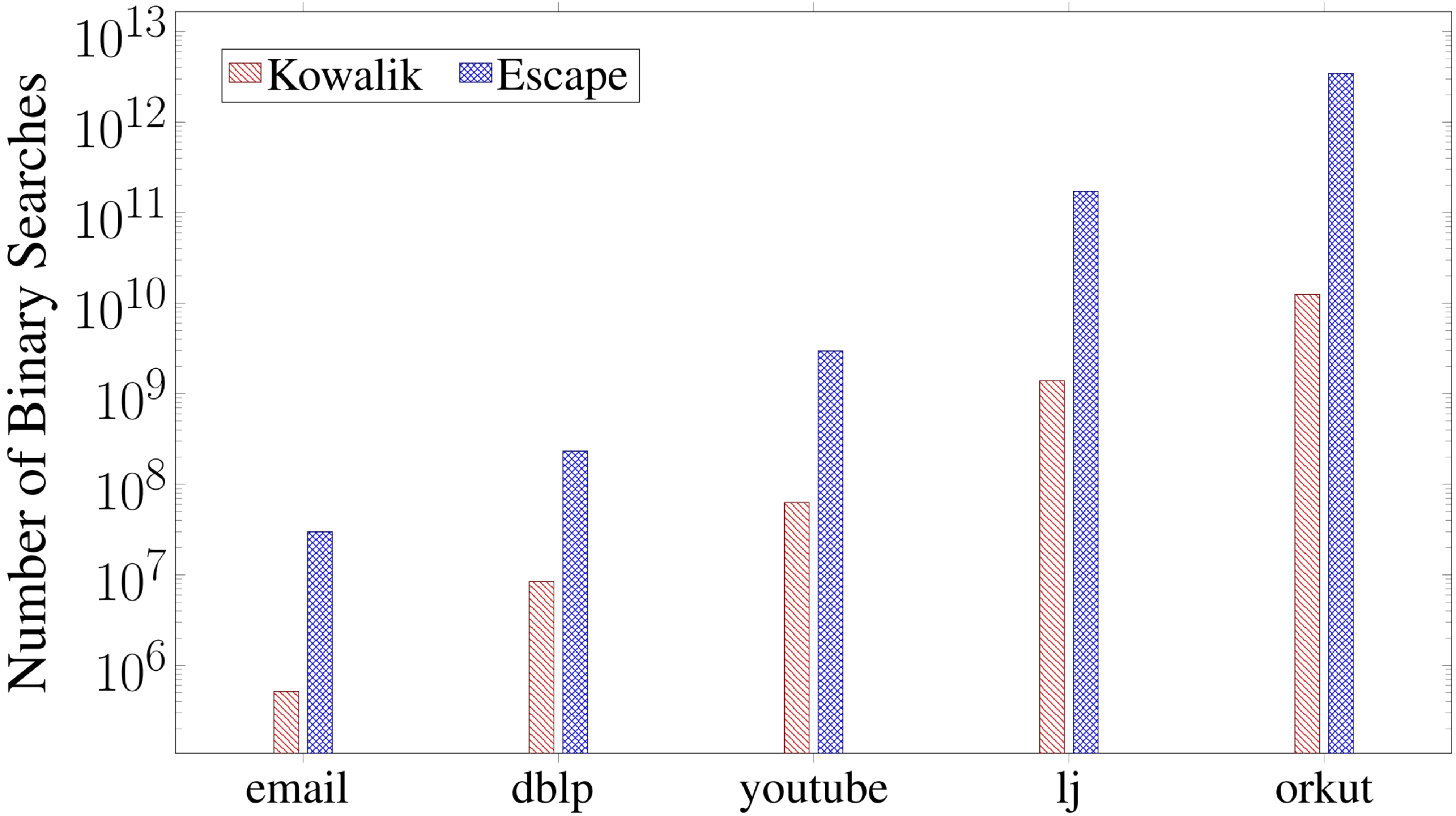
Main Running Times



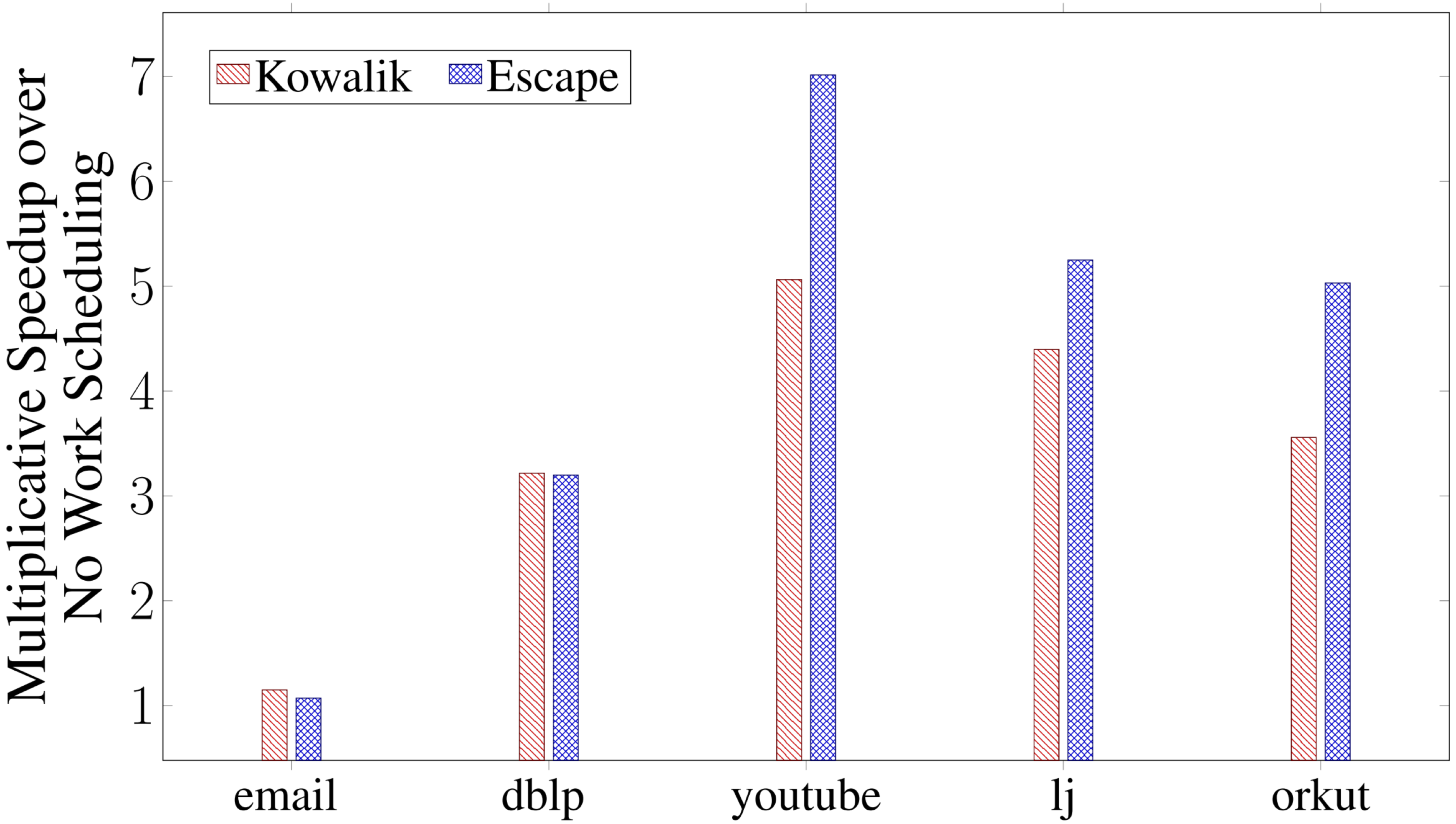
- Up to 32.2x speedups over best sequential implementation
- Up to 818.12x speedups over ESCAPE package
- 8417.3 s on friendster graph

36h-Thread Kowalik: (0.0025 s) (0.014 s) (0.33 s) (5.85 s) (136.98 s)
(fastest)

Binary Searches in Kowalik vs ESCAPE



Work Scheduling



Conclusion

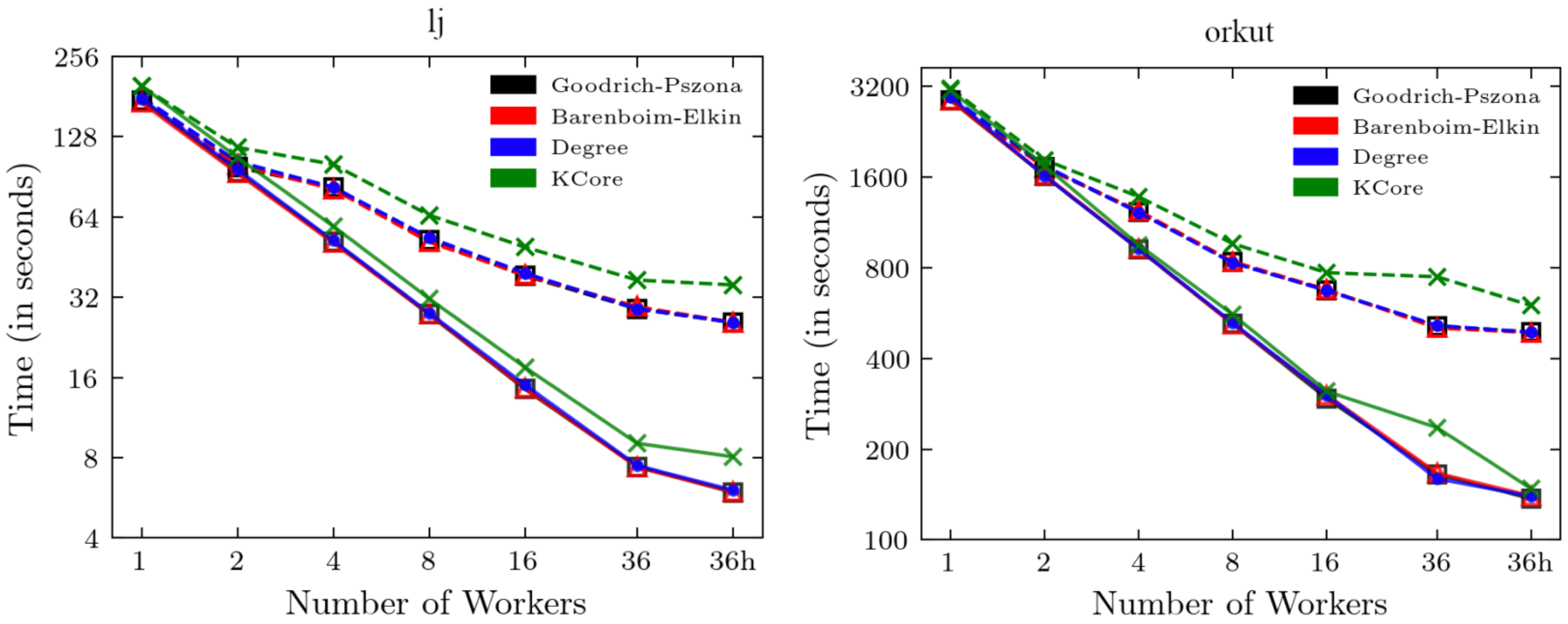
Conclusion

- New parallel algorithms for five-cycle counting
- Strong theoretical bounds + fast performance

- Github: https://github.com/ParAlg/gbbs/tree/master/benchmarks/CycleCounting

Thank You

Scalability of Parallel Kowalik



Dashed line = With work scheduling
Solid line = Without work scheduling