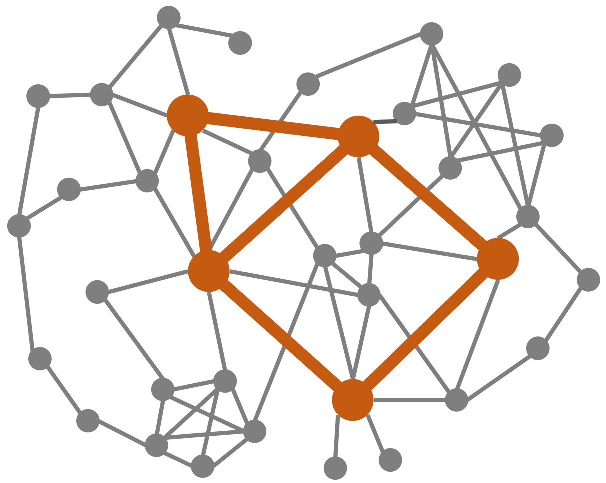
AutoMine

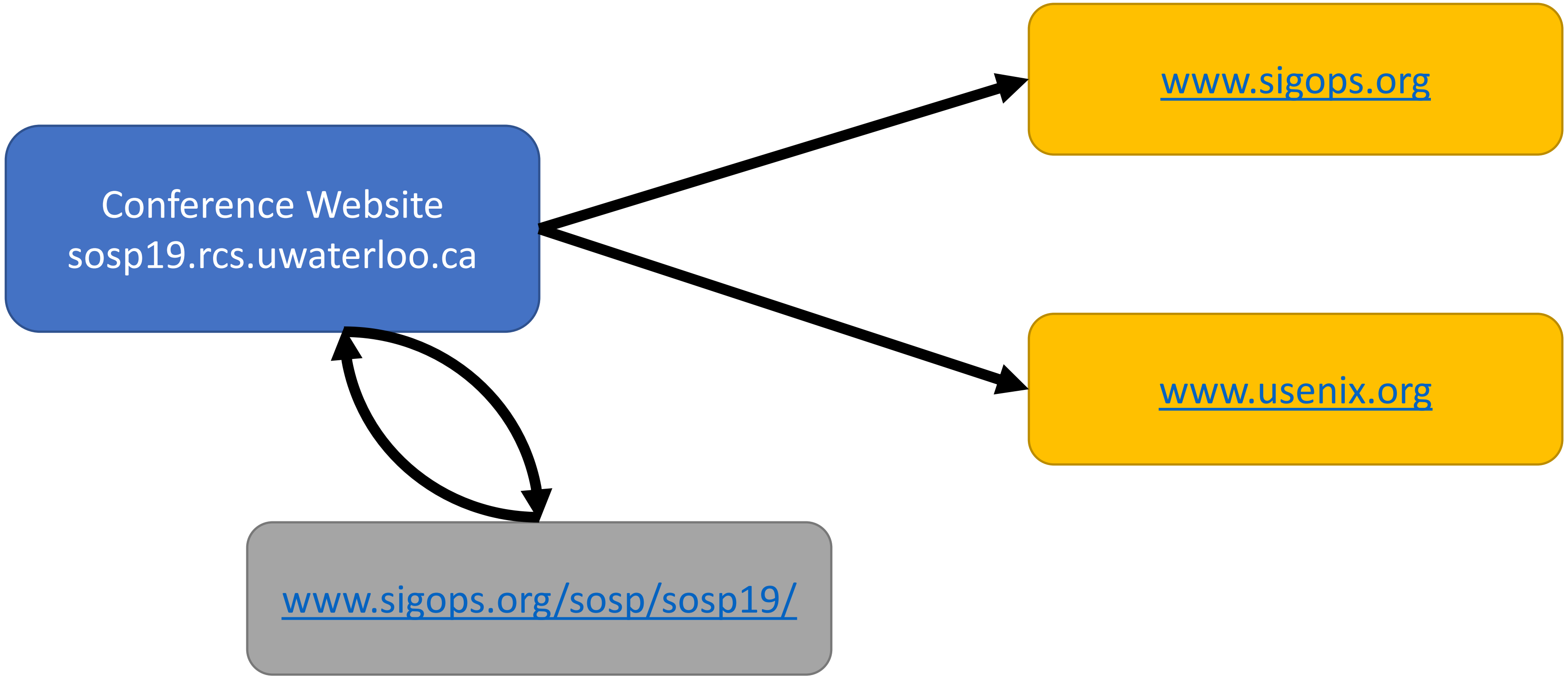
Harmonizing High-Level Abstraction and High Performance for Graph Mining

Daniel Mawhirter, Bo Wu

SOSP, October 30, 2019



Graphs



- But the internet is big! (And so are other graph datasets)

Big Graphs

- 2 Billion Facebook users
- 3 Billion base pairs in human genome
- 20 Billion internet connected devices
- Trillions of connections between them

- Many graph processing systems are designed to optimize graph traversal problems
  - PowerGraph [OSDI'12], GraphChi [OSDI'12], GraphX [OSDI'14], X-Stream [SOSP'13]
- Running BFS on Friendster in X-Stream takes 15s for just a linear-time traversal

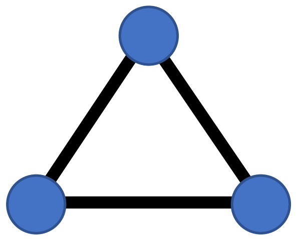
Graph Mining

- Aims to discover *structural patterns* in a graph

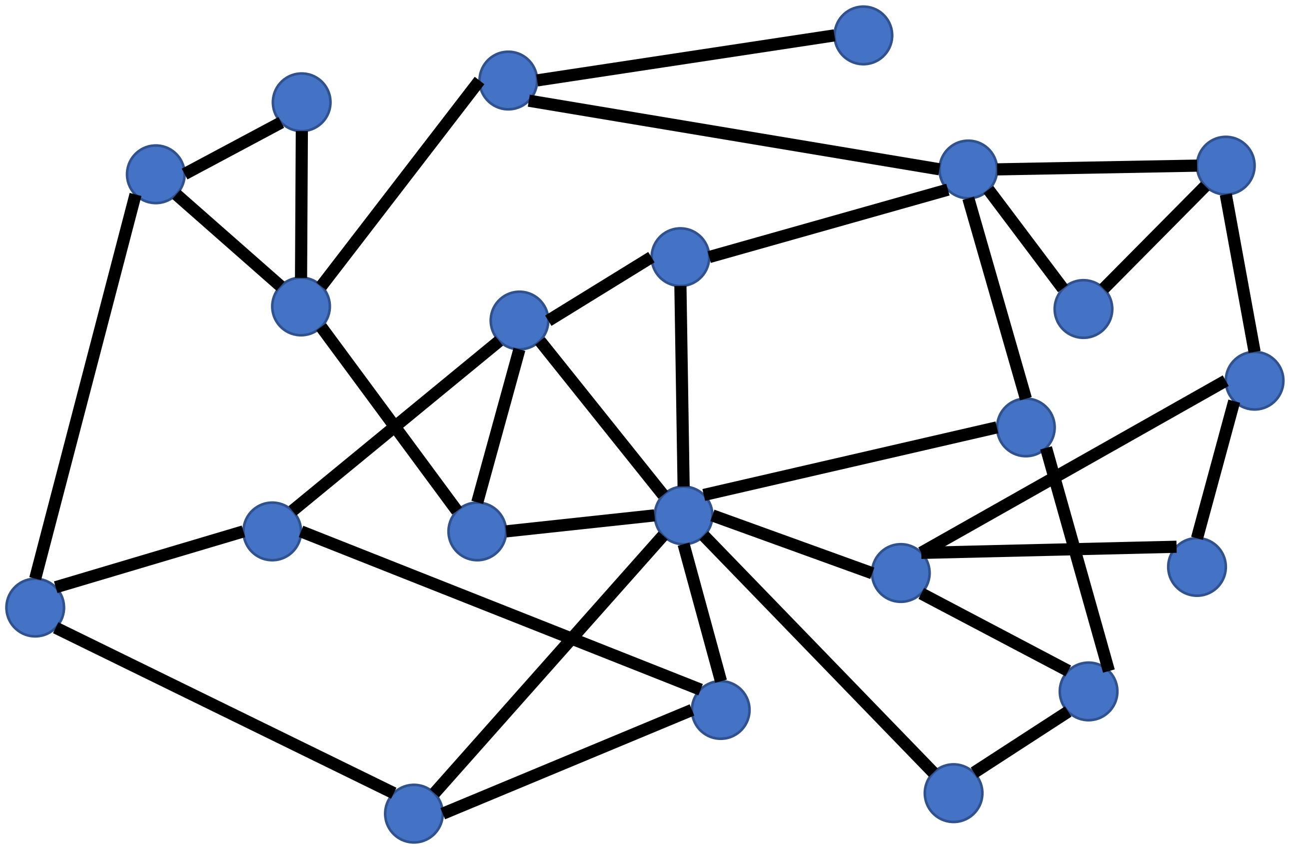
- Examples:
  - Motif Counting finds all subgraphs of a given size
  - Frequent Subgraph Mining uses labels to further distinguish patterns

- Useful in anomaly/fraud detection, bioinformatics, large scale graph comparison

Triangle Counting

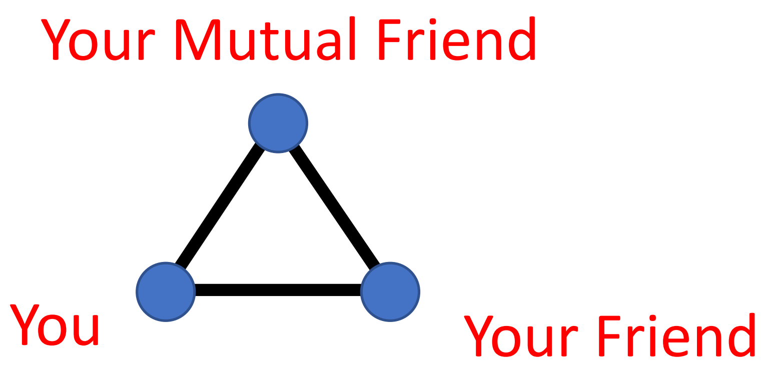


Pattern

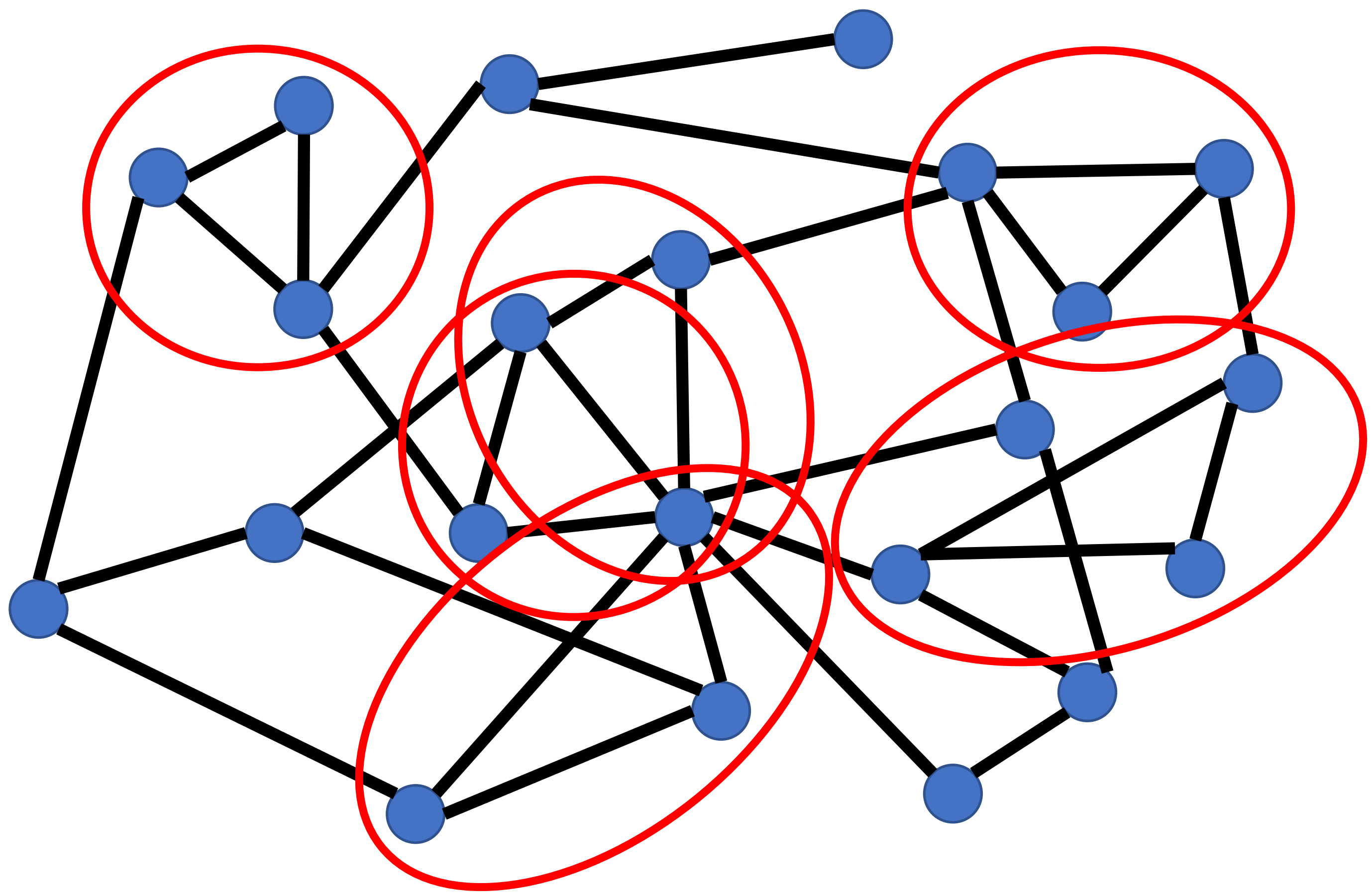


Dataset

Triangle Counting

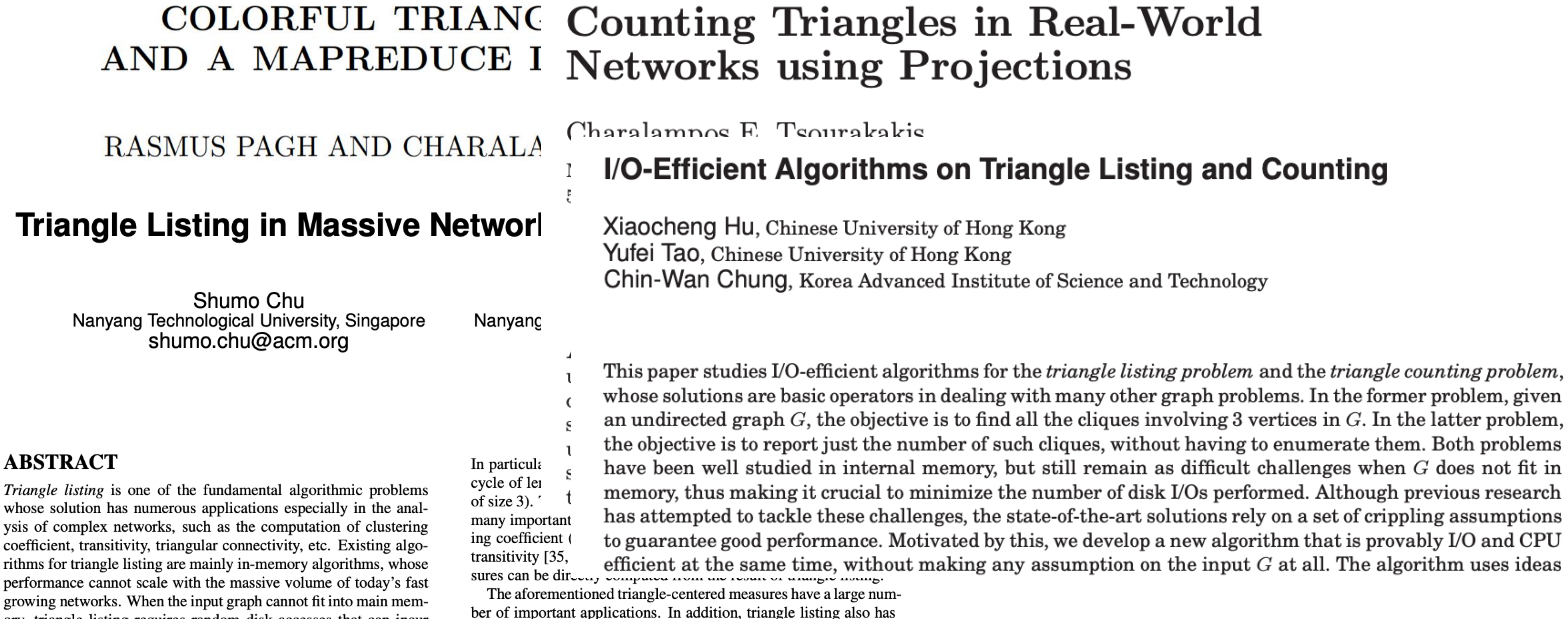


Pattern



Dataset

Triangle Counting is well-studied



COLORFUL TRIANG
AND A MAPREDUCE I

RASMUS PAGH AND CHARALA

Triangle Listing in Massive Networl

Shumo Chu
Nanyang Technological University, Singapore
shumo.chu@acm.org

Nanyang

ABSTRACT

Triangle listing is one of the fundamental algorithmic problems whose solution has numerous applications especially in the analysis of complex networks, such as the computation of clustering coefficient, transitivity, triangular connectivity, etc. Existing algorithms for triangle listing are mainly in-memory algorithms, whose performance cannot scale with the massive volume of today's fast growing networks. When the input graph cannot fit into main mem-

In particula
cycle of le
of size 3).
many important
ing coefficient
transitivity [35,
sures can be dir
The aforementioned triangle-centered measures have a large number of important applications. In addition, triangle listing also has

Counting Triangles in Real-World Networks using Projections

Charalampos E. Tsourakakis

I/O-Efficient Algorithms on Triangle Listing and Counting

Xiaocheng Hu, Chinese University of Hong Kong
Yufei Tao, Chinese University of Hong Kong
Chin-Wan Chung, Korea Advanced Institute of Science and Technology

This paper studies I/O-efficient algorithms for the *triangle listing problem* and the *triangle counting problem*, whose solutions are basic operators in dealing with many other graph problems. In the former problem, given an undirected graph G, the objective is to find all the cliques involving 3 vertices in G. In the latter problem, the objective is to report just the number of such cliques, without having to enumerate them. Both problems have been well studied in internal memory, but still remain as difficult challenges when G does not fit in memory, thus making it crucial to minimize the number of disk I/Os performed. Although previous research has attempted to tackle these challenges, the state-of-the-art solutions rely on a set of crippling assumptions to guarantee good performance. Motivated by this, we develop a new algorithm that is provably I/O and CPU efficient at the same time, without making any assumption on the input G at all. The algorithm uses ideas

Triangle Counting is well-studied

GraphChallenge

Motivation | Challenges | Data Sets | Scenarios | Submit | News | Champions | Contact

Home

Scenarios

Notional Scenarios for the 2017 HIVE Graph Challenge

In this era of big data, the rates at which these data sets grow continue to accelerate. The ability to manage and analyze the largest data sets is always severely taxed. The most challenging of these data sets are those containing relational or network data. The HIVE challenge is envisioned to be an annual challenge that will advance the state of the art in graph analytics on extremely large data sets. The primary focus of the challenges will be on the expansion and acceleration of graph analytic algorithms through improvements to algorithms and their implementations, and especially importantly, through special purpose hardware such as distributed and grid computers, and GPUs. Potential approaches to accelerate graph analytic algorithms include such methods as massively parallel computation, improvements to memory utilization, more efficient communications, and optimized data processing units.

The 2017 HIVE challenge is composed of two challenges: the first focuses on subgraph isomorphism and the second on community detection. The baseline algorithms for the first challenge are recently developed algorithms that find triangles and k-trusses (J. Wang 2012). The triangle counting algorithms can be considered as a special case of subgraph isomorphism where the subgraph of interest is restricted to a triangle. Although these algorithms do not find matching subgraphs of a general description, they can be used as components in algorithms that do. K-truss search algorithms can potentially support subgraph isomorphism algorithms through the characterization of a larger graph

COL
AND A

RASMUS

Triangle Listin

Shur
Nanyang Technologic
shumo.ch

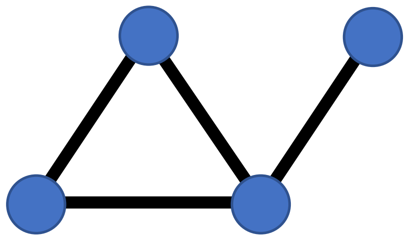
ABSTRACT

Triangle listing is one of the funda
whose solution has numerous applic
ysis of complex networks, such as
coefficient, transitivity, triangular co
rithms for triangle listing are mainly
performance cannot scale with the n
growing networks. When the input gr

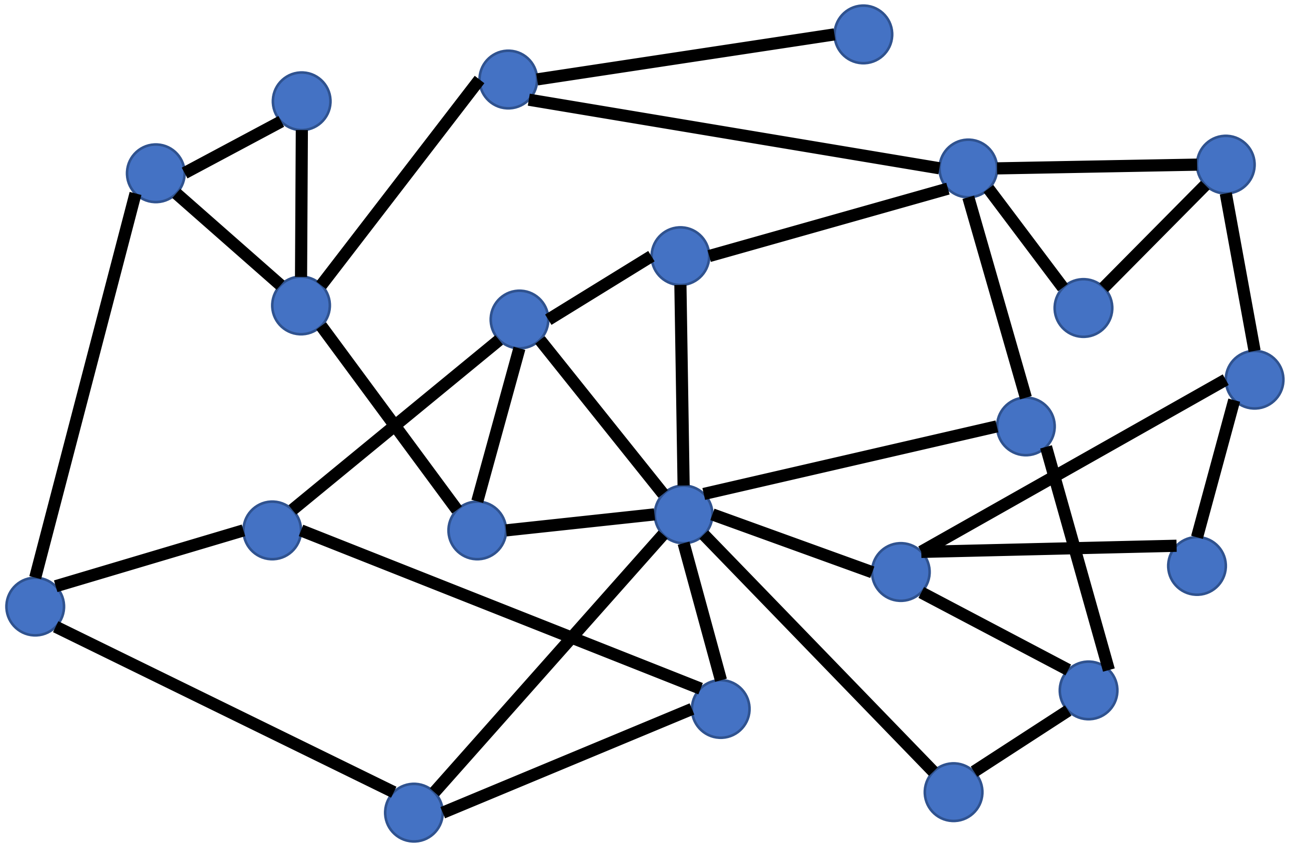
unting

e *triangle counting problem*,
In the former problem, given
s in G. In the latter problem,
nerate them. Both problems
nges when G does not fit in
Although previous research
set of crippling assumptions
hat is provably I/O and CPU
ll. The algorithm uses ideas

What about other patterns?

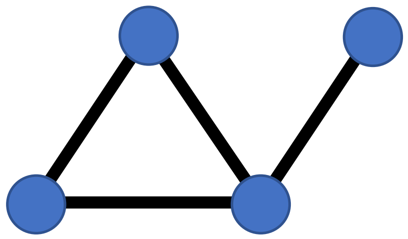


Pattern

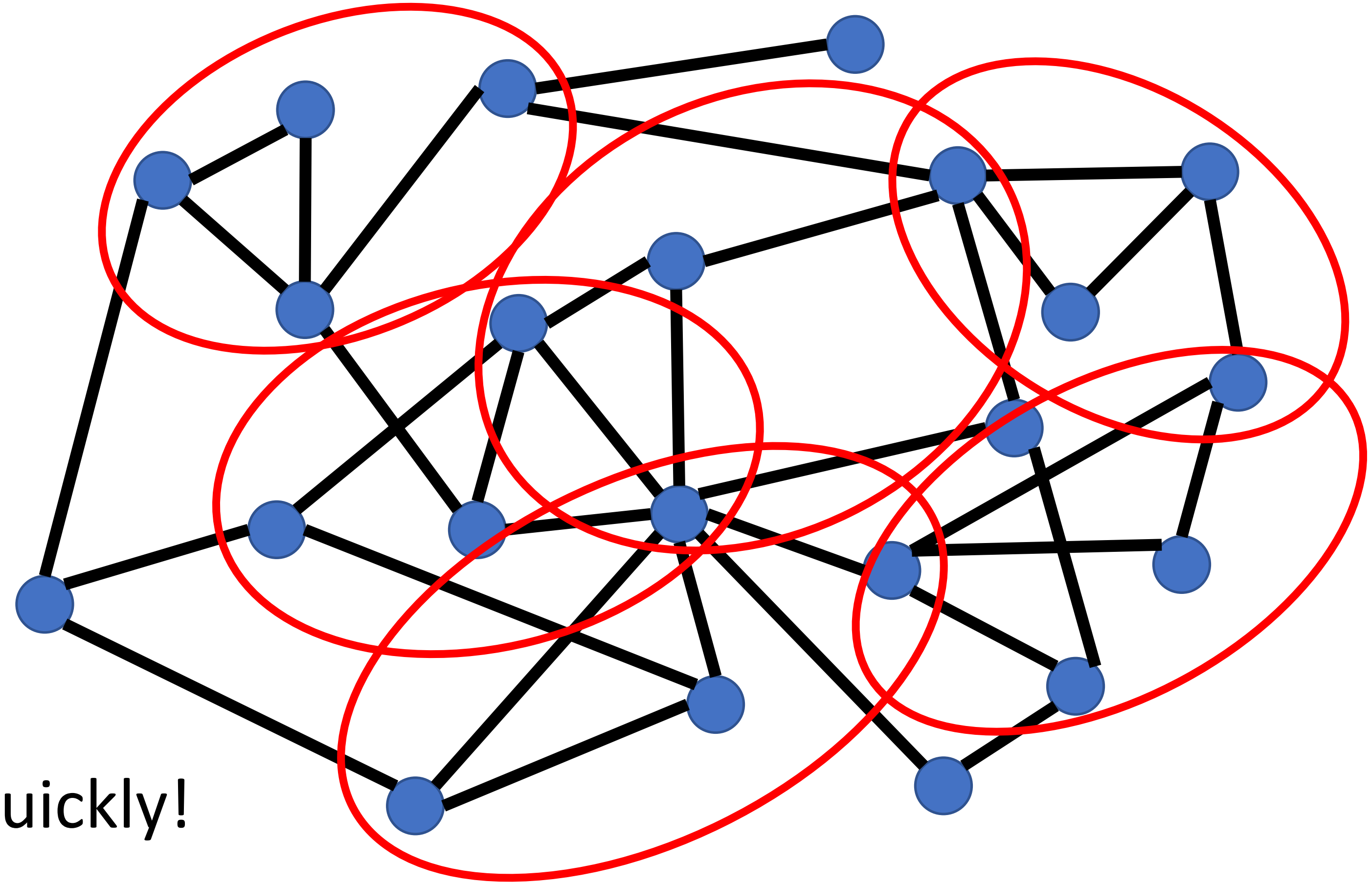


Dataset

What about other patterns?



Pattern

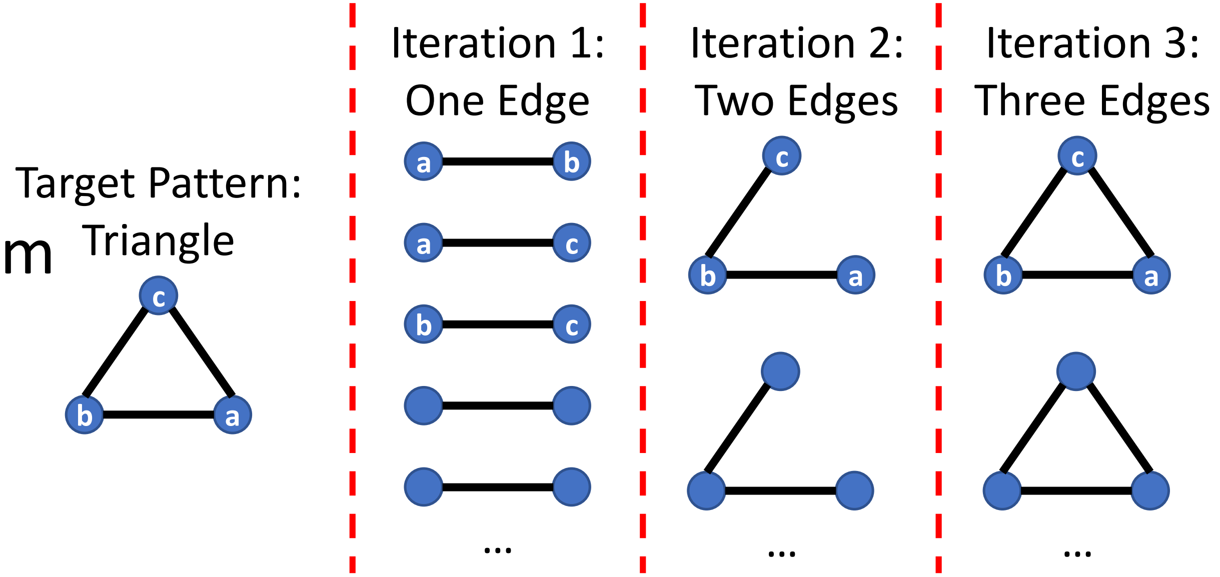


Dataset

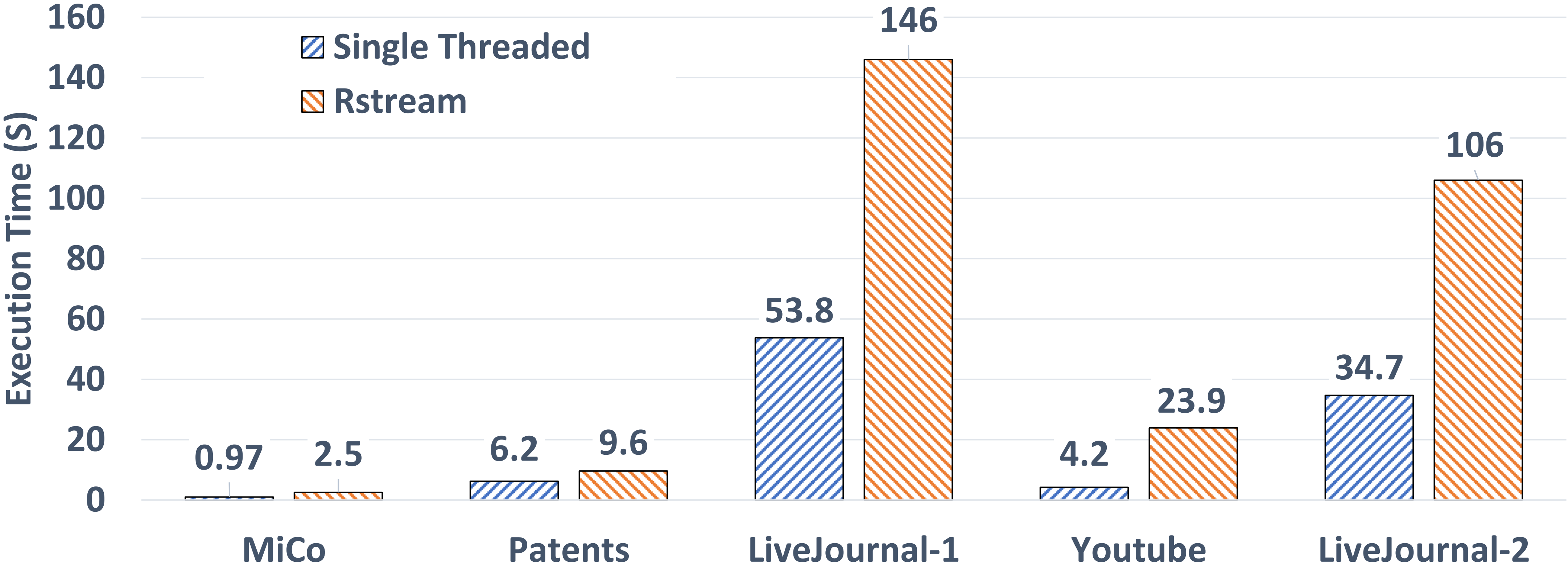
- Things get complicated quickly!

Prior General Mining Systems

- Arabesque[SOSP'15] and RStream[OSDI'18] are two state-of-the-art graph mining systems
- Idea: Enumerate the embeddings (i.e., subgraph instances) and run isomorphism tests
- Arabesque is a distributed system that implements an embedding-centric interface
- RStream runs on a single-machine and supports disk-streaming

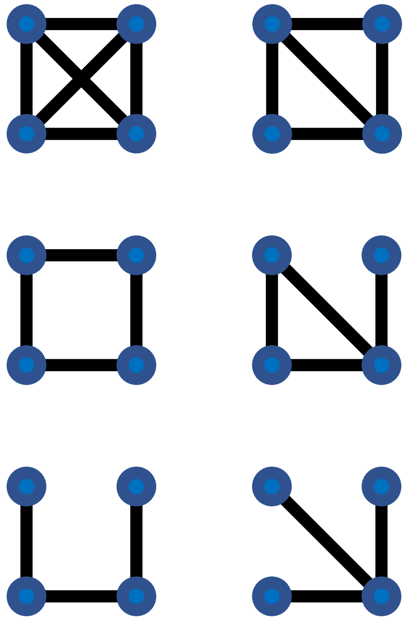


Single Thread Comparison



AutoMine

- First of its kind topological compiler for graph mining
- Automates the manual algorithm design process



$$
\begin{gathered}
A \in \boldsymbol{V} \\
B_A \in Adj(A) \\
C_{AB} \in Adj(A) \cap Adj(B) \\
\dots \\
instances[clique_4] \mathrel{+}= D_{ABC} \\
instances[rectangle] \mathrel{+}= D_{BC}
\end{gathered}
$$

Algorithms

Dataset

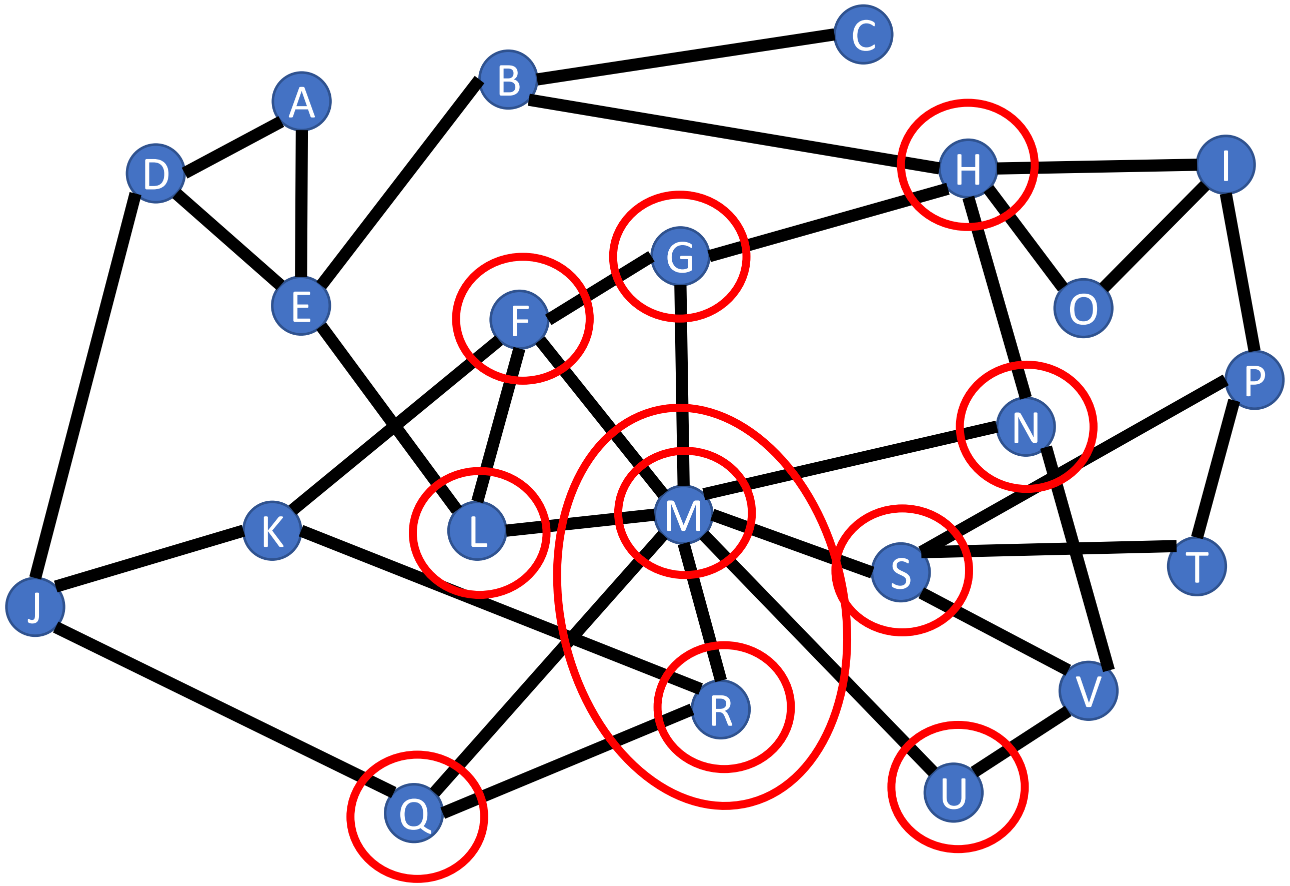
$$
\begin{aligned}
&for(A : \boldsymbol{V})\ \{ \\
&\quad for\big(B_A : Adj(A)\big)\ \{ \\
&\quad\quad for(C_{AB} : Adj(A)\ \dots)\ \{ \\
&\quad\quad\quad \dots \\
&\quad \} \\
&\}
\end{aligned}
$$

Results!

Systems

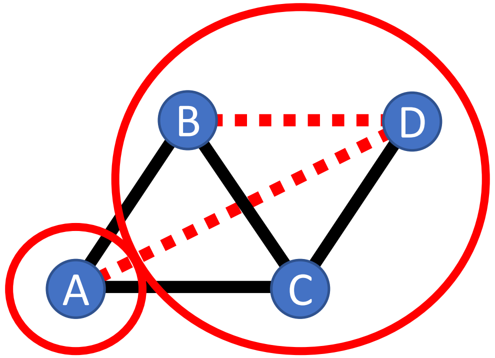
Techniques

- Set Modeling
- Vertex M
- Adjacent(M)
- $R \in Adj(M)$



Techniques

- Set Operations
- Begin from vertex A
- Discover vertices B-D
- Insert missing edges to encode all relationships
- Intersection (∩) and Difference (−) are sufficient, proof in paper



$$A$$

$$B \in Adj(A)$$

$$\cancel{C \in Adj(A), C \in Adj(B)}$$

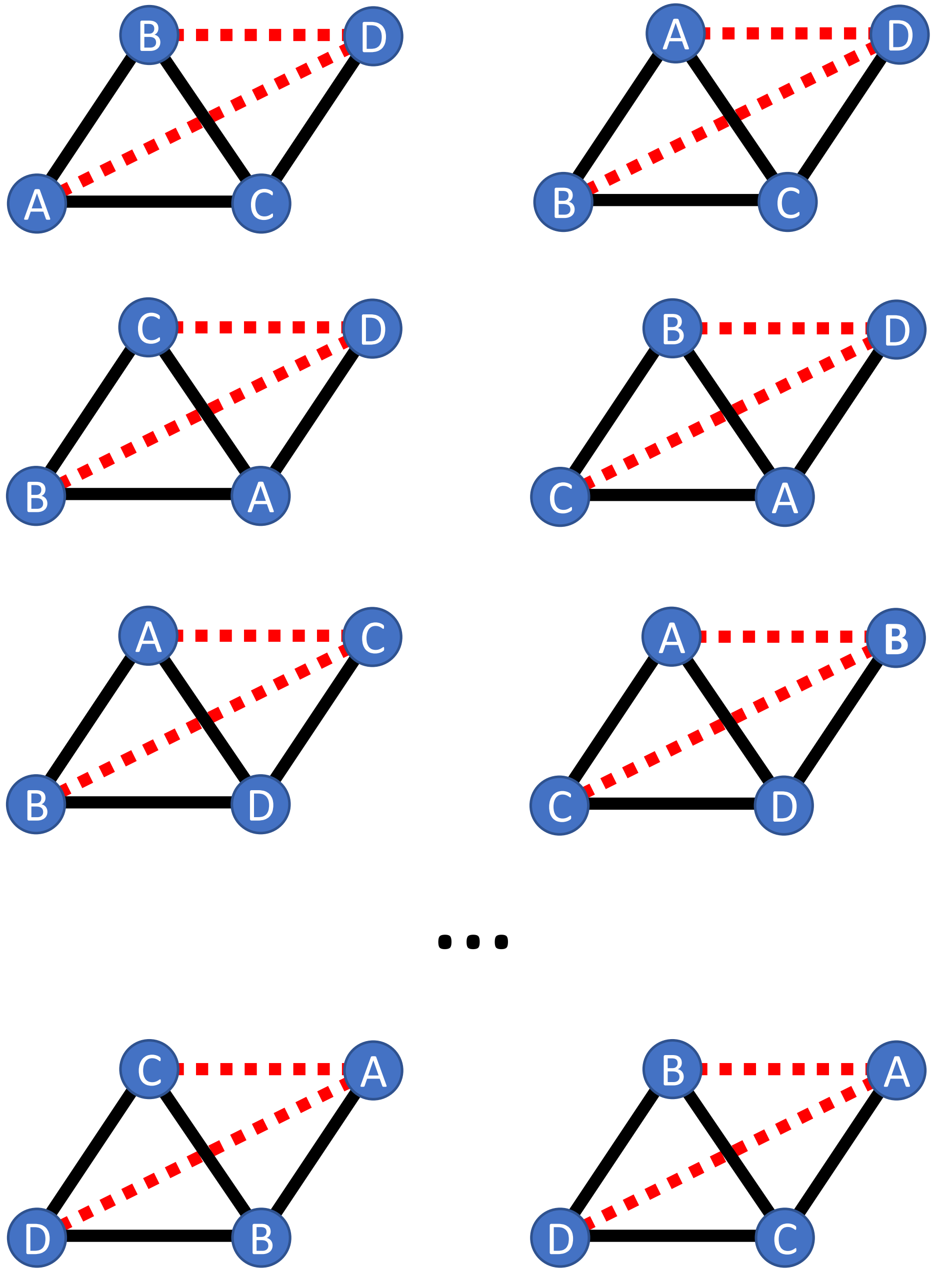
$$C \in Adj(A) \cap Adj(B)$$

$$\cancel{D \in Adj(C), D \notin Adj(A), D \notin Adj(B)}$$

$$D \in Adj(C) - Adj(A) - Adj(B)$$

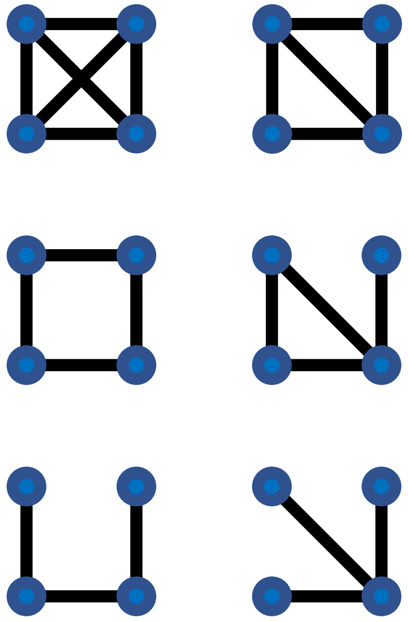
Techniques

- Scheduling space (permutations)
- Different orders imply different order of operations
- All are correct, just with different performance implications
- Choice of order is described in the paper



AutoMine

- First of its kind topological compiler for graph mining
- Automates the manual algorithm design process



$$A \in \boldsymbol{V}$$
$$B_A \in Adj(A)$$
$$C_{AB} \in Adj(A) \cap Adj(B)$$
$$\ldots$$
$$instances[clique_4] \mathrel{+}= D_{ABC}$$
$$instances[rectangle] \mathrel{+}= D_{BC}$$

Algorithms

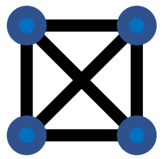
Dataset

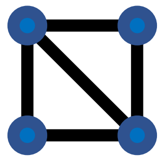
$for(A : \boldsymbol{V})\ \{$
$\quad for\big(B_A : Adj(A)\big)\ \{$
$\quad\quad for(C_{AB} : Adj(A)\ \ldots)\ \{$
$\quad\quad\quad \ldots$
$\quad\ \}$
$\}$

Results!

Systems

Map to low-level code

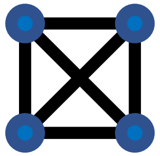


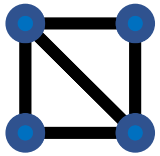


$$A \in \boldsymbol{V}$$
$$B_A \in Adj(A)$$
$$C_{AB} \in Adj(A) \cap Adj(B_A)$$
$$D_{ABC} \in Adj(A) \cap Adj(B_A) \cap Adj(C_{AB})$$
$$instances[clique\_4] += D_{ABC}$$

$$A \in \boldsymbol{V}$$
$$B_A \in Adj(A)$$
$$C_{AB} \in Adj(A) \cap Adj(B_A)$$
$$D_{BC} \in Adj(B_A) \cap Adj(C_{AB}) - Adj(A)$$
$$instances[chordal] += D_{BC}$$

Map to low-level code

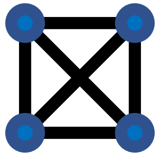
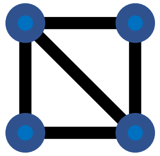




$$
\begin{gathered}
A \in \boldsymbol{V} \\
B_A \in Adj(A) \\
C_{AB} \in Adj(A) \cap Adj(B_A) \\
D_{ABC} \in C_{AB} \cap Adj(C_{AB}) \\
instances[clique\_4] \mathrel{+}= D_{ABC}
\end{gathered}
$$

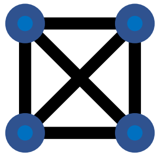
$$
\begin{gathered}
A \in \boldsymbol{V} \\
B_A \in Adj(A) \\
C_{AB} \in Adj(A) \cap Adj(B_A) \\
C_B \in Adj(B_A) - Adj(A) \\
D_{BC} \in C_B \cap Adj(C_{AB}) \\
instances[chordal] \mathrel{+}= D_{BC}
\end{gathered}
$$

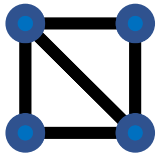
Map to low-level code

$$A \in \boldsymbol{V}$$
$$B_A \in Adj(A)$$
$$C_{AB} \in Adj(A) \cap Adj(B_A)$$
$$C_B \in Adj(B_A) - Adj(A)$$
$$D_{ABC} \in C_{AB} \cap Adj(C_{AB})$$
$$D_{BC} \in C_B \cap Adj(C_{AB})$$
$$instances[clique\_4] \mathrel{+}= D_{ABC}$$
$$instances[chordal] \mathrel{+}= D_{BC}$$

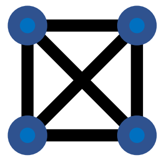
Map to low-level code

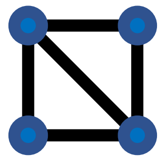




```
for v0 in V:
  for v1 in Adj(A):
    y0y1 = Adj(v0) ∩ Adj(v1)
    n0y1 = Adj(v1) - Adj(v0)
    for v2 : y0y1:
      y0y1y2 = y0y1 ∩ Adj(v2)
      n0y1y2 = n0y1 ∩ Adj(v2)
      counter_0 += y0y1y2.size()
      counter_1 += n0y1y2.size()
```

Map to low-level code





```
Graph g(file);
#pragma omp parallel for
for(vidType v0 = 0; v0 < n_vertices; v_0++) {
  for(vidType v1 : g.Adj(v0)) {
    VertexSet y0y1 = g.Adj(v0) & g.Adj(v1);
    VertexSet n0y1 = g.Adj(v1) - g.Adj(v0);
    for(vidType v2 : y0y1) {
      VertexSet y0y1y2 = y0y1 & Adj(v2);
      VertexSet n0y1y2 = n0y1 & Adj(v2);
      record_0(v0, v1, v2, y0y1y2);
      record_1(v0, v1, v2, n0y1y2);
    }
  }
}
```

Parallelization

Data Reuse

VertexSets no longer needed once they go out of scope

API (Automating the Whole Process)

Basic APIs:

$Pattern\ definePattern(Edge[]\ edgelist);$

$Program\ countPatterns(Pattern[]\ patterns);$

$Program\ enumeratePatterns(Pattern[]\ patterns);$

Application-Level APIs:

$Program\ CC(int\ size);$

$Program\ MC(int\ size);$

$Program\ FSM(int\ size, int\ support);$

Evaluation

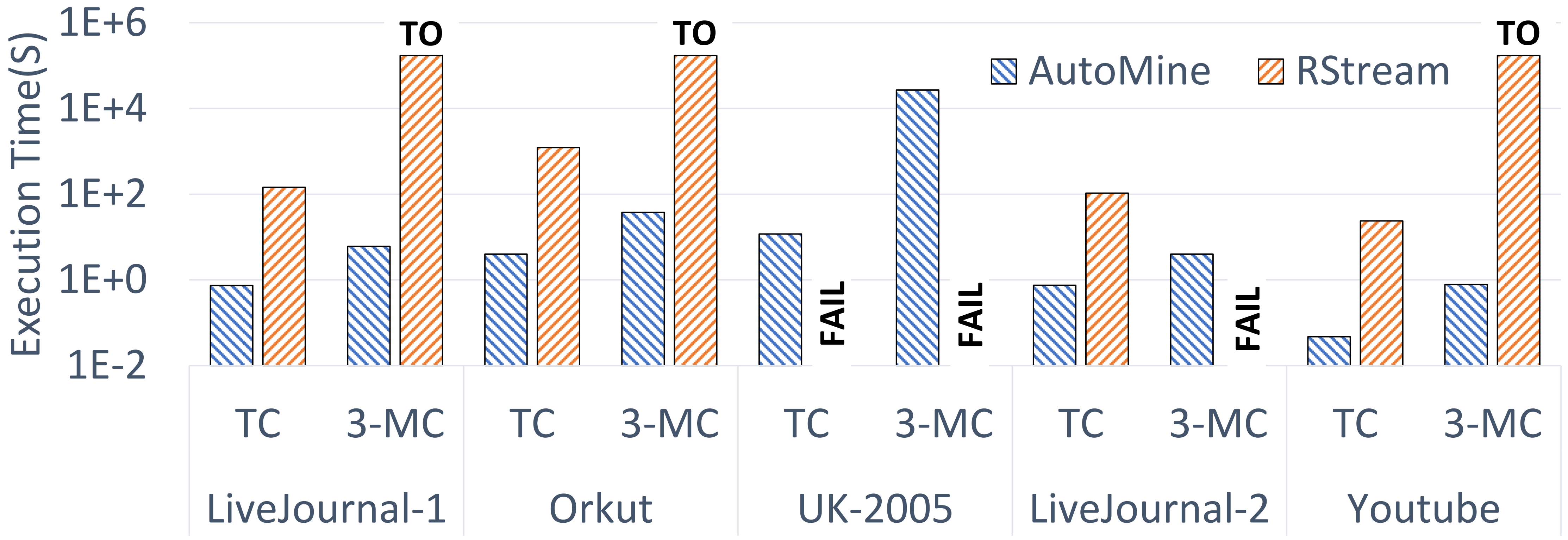
- 2x 10-core Intel Xeon E5-2630 v4 CPUs (40 threads), 64gb memory

Graph	Vertices	Edges	Domain
CiteSeer	3264	4536	Publication citation
MiCo	96638	1080156	Co-authorship
Patents	3.8M	16.5M	US Patents
LiveJournal-1	4.8M	42.9M	Social network
Orkut	3.1M	117.2M	Social network
UK-2005	39.5M	783M	Web graph
Youtube	1.1M	3M	Social network
LiveJournal-2	4M	34.7M	Social network
GSH-2015	988.5M	25.7B	Web graph

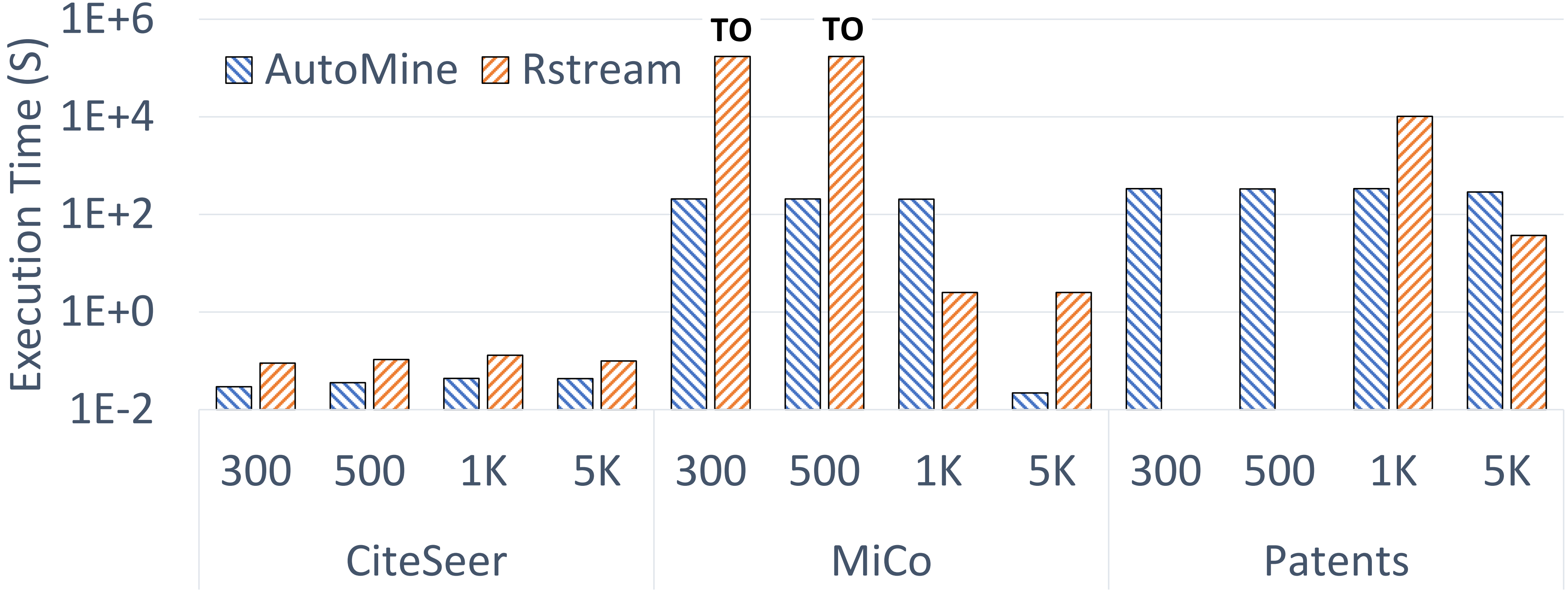
Performance (Size 3)

		CiteSeer	MiCo	Patents
Triangle Counting	AutoMine	0.01	0.04	0.14
	RStream	0.01	2.5	9.6
	Arabesque	38.1	43.1	114.9
Motif	AutoMine	0.016	0.12	0.5
	RStream	0.13	1666.9	1149.1
	Arabesque	40.6	51.7	116
Frequent Subgraph 5k	AutoMine	0.02	0.039	3.9
	RStream	0.087	2.54	36.3
	Arabesque	41.6	120.8	F

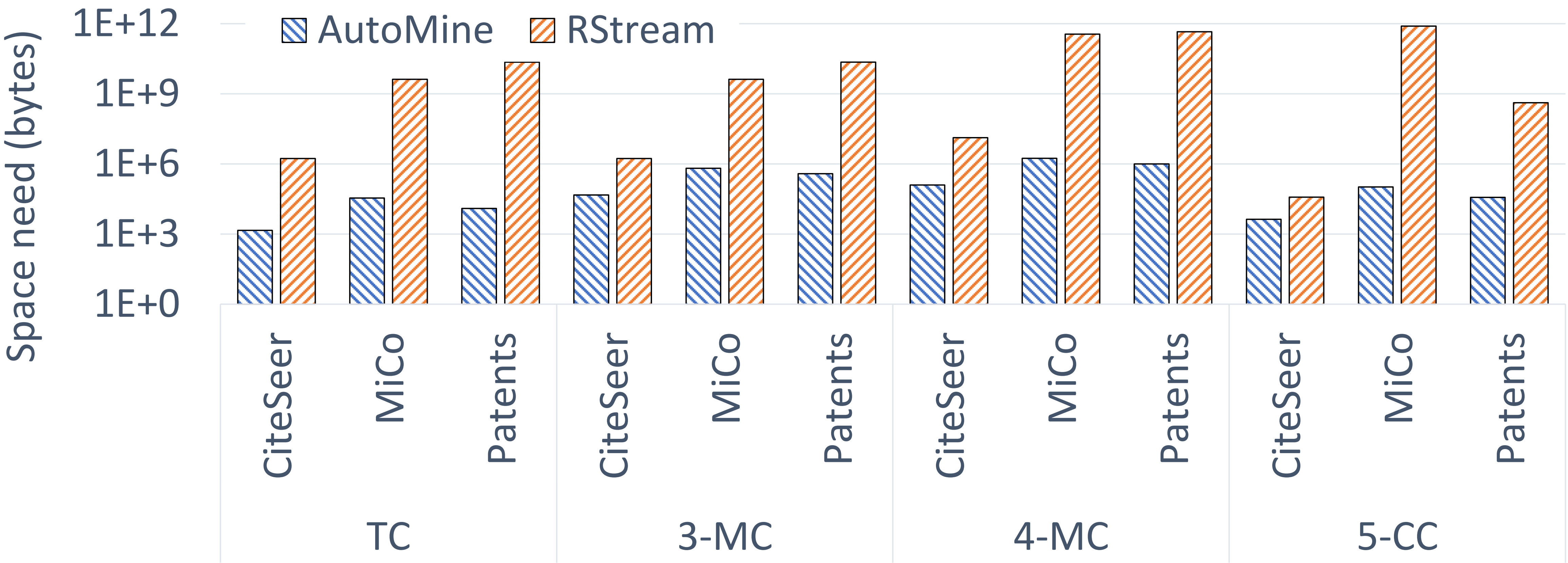
Performance vs Rstream (Larger Graphs)



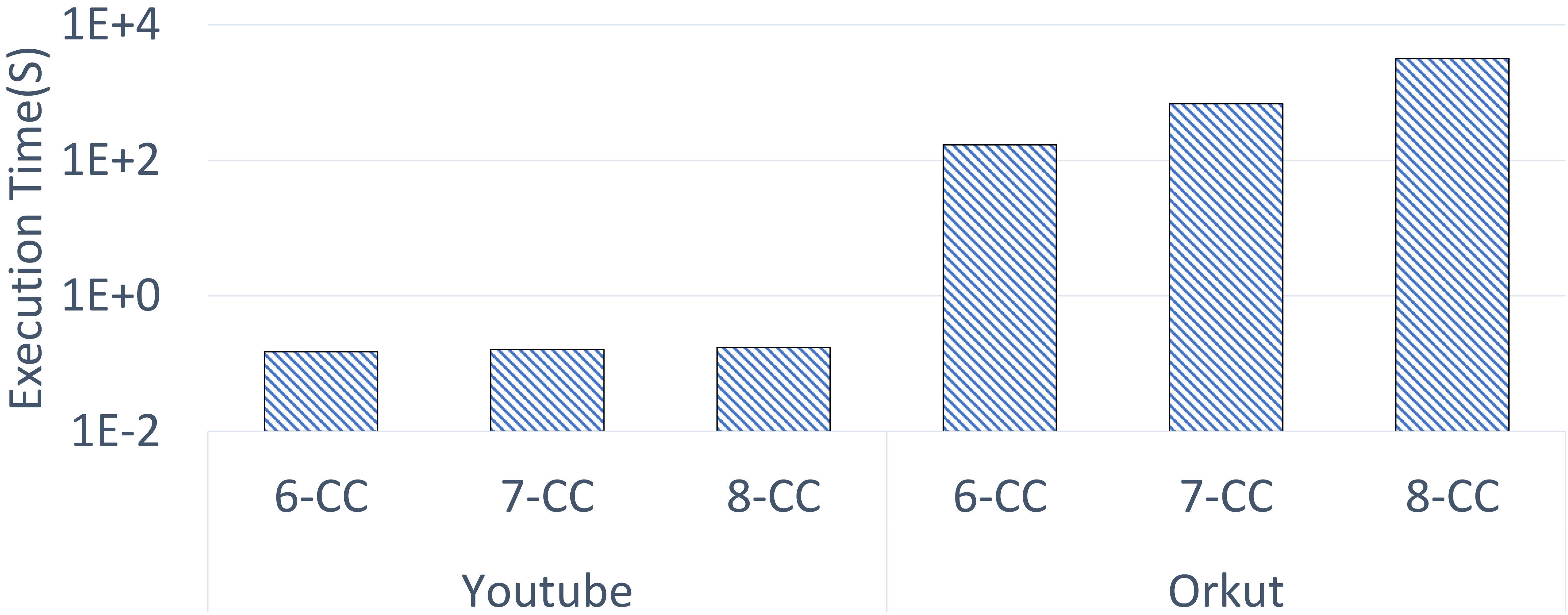
Performance vs Rstream (FSM-4)



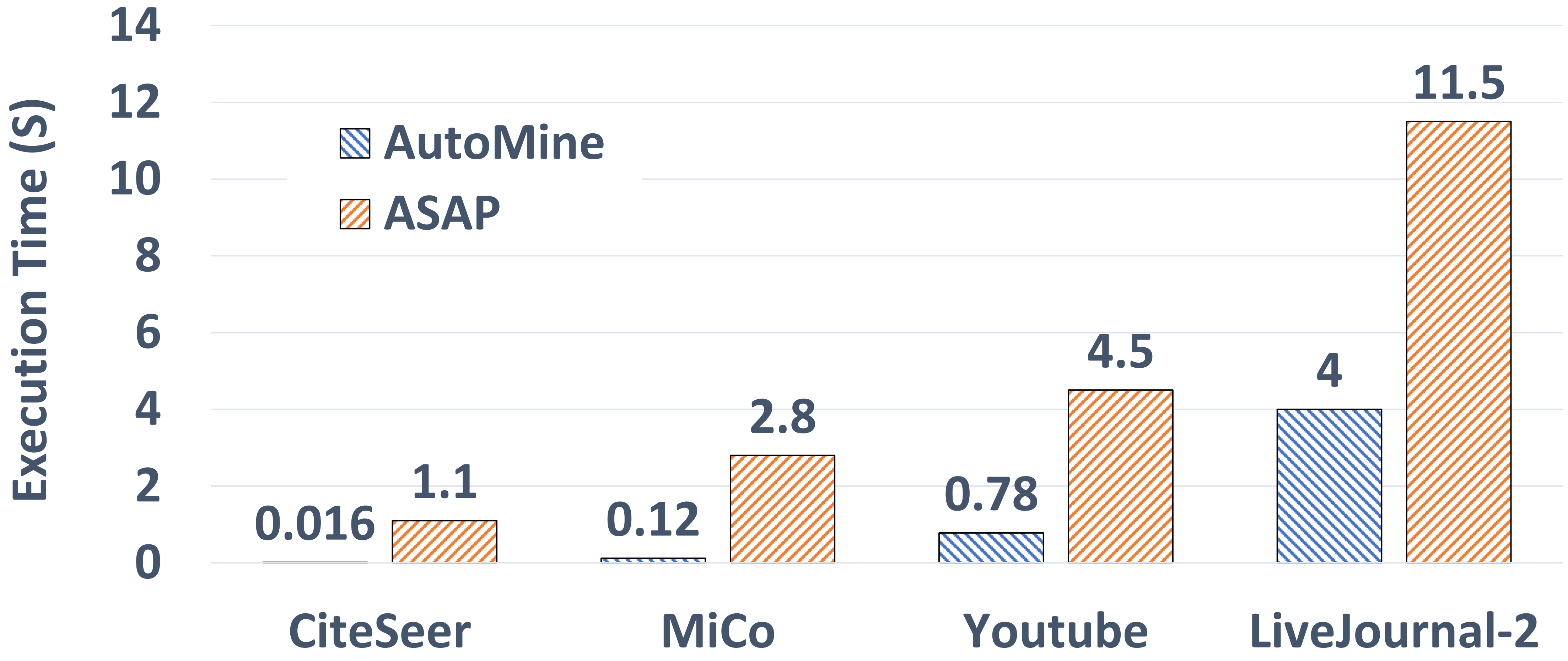
Intermediate Data



Performance (Large Cliques)



Performance vs ASAP [OSDI'18]



Conclusions

- Manual algorithms may be much faster than graph mining systems
- Manual algorithm design doesn’t scale to larger patterns
- AutoMine harmonizes the high-level abstraction and high performance for graph mining through automated algorithm and code generation
- Can we extend this idea to other domains?

AutoMine

Daniel Mawhirter
dmawhirt@mymail.mines.edu

Bo Wu
bwu@mines.edu

COLORADO SCHOOL OF MINES