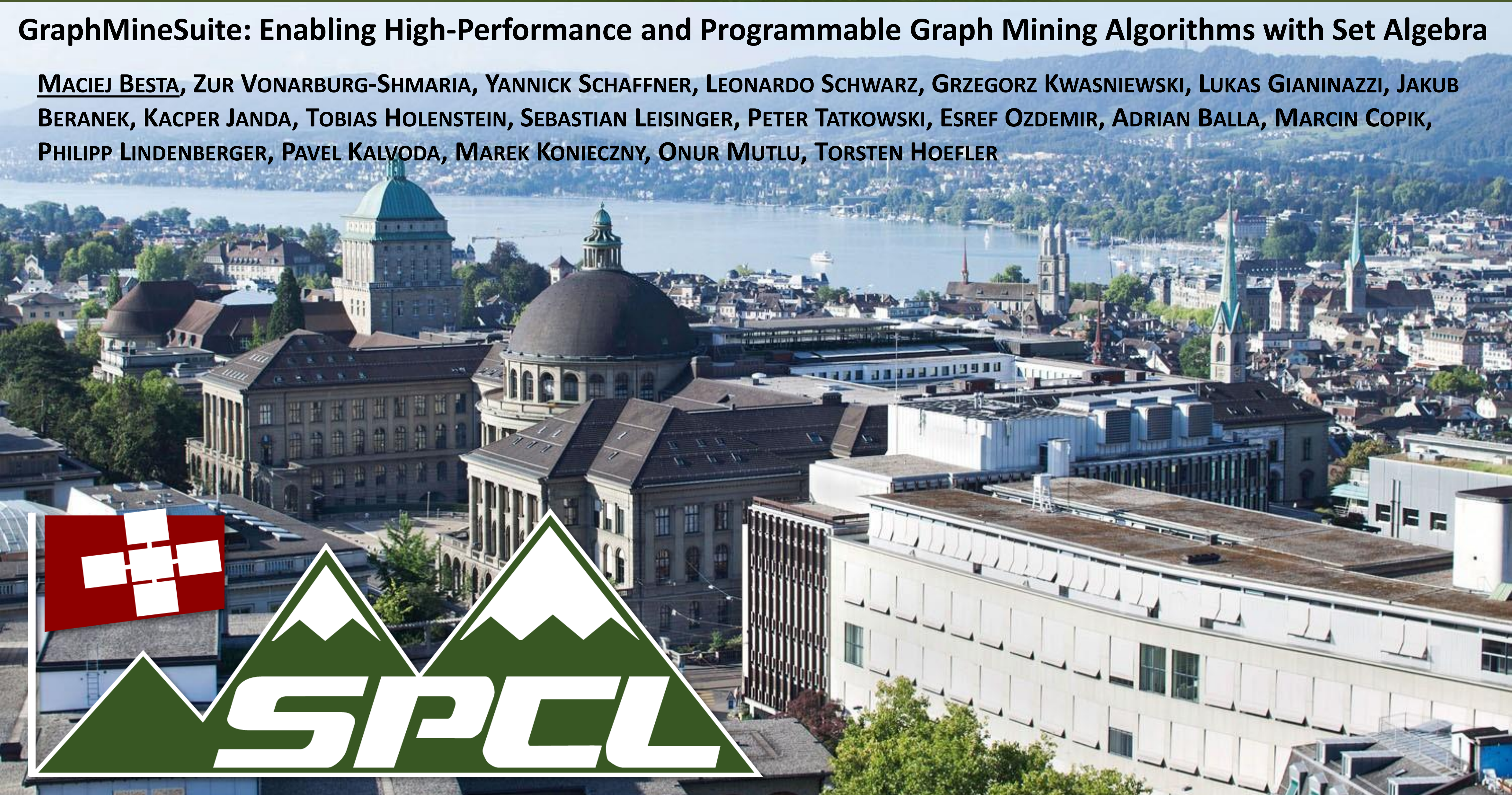
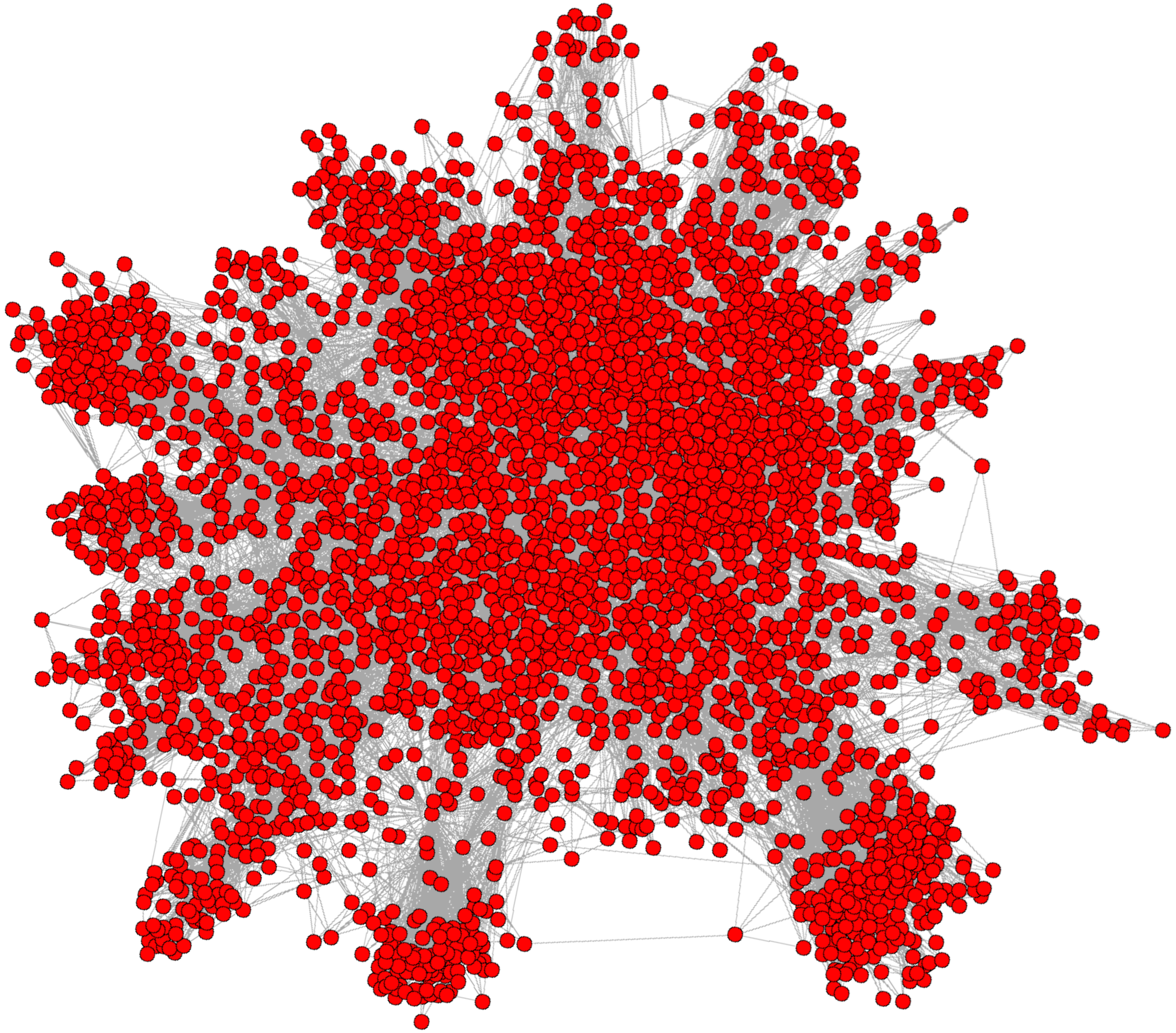
GraphMineSuite: Enabling High-Performance and Programmable Graph Mining Algorithms with Set Algebra

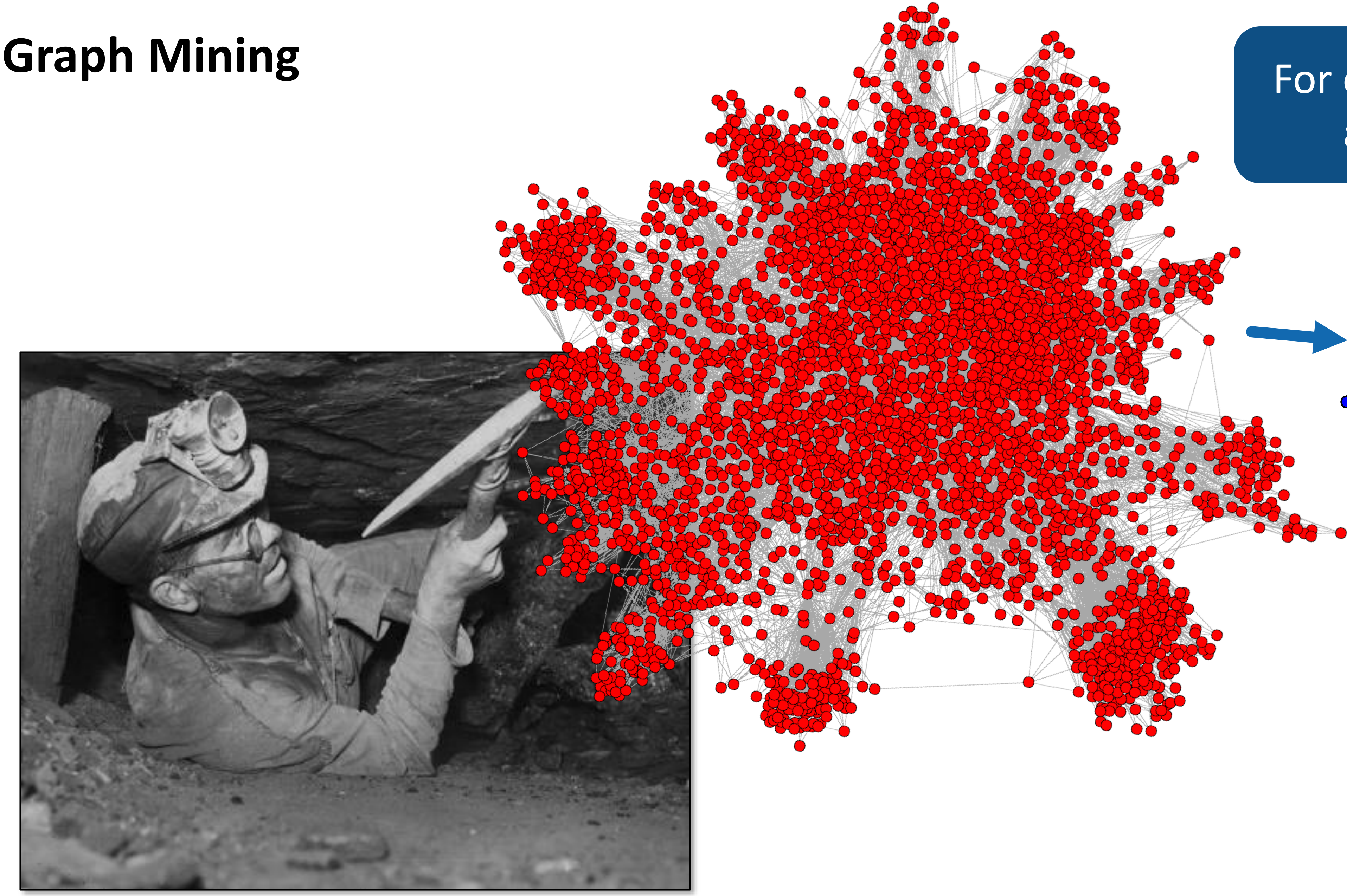
Maciej Besta, Zur Vonarburg-Shmaria, Yannick Schaffner, Leonardo Schwarz, Grzegorz Kwasniewski, Lukas Gianinazzi, Jakub Beranek, Kacper Janda, Tobias Holenstein, Sebastian Leisinger, Peter Tatkowski, Esref Ozdemir, Adrian Balla, Marcin Copik, Philipp Lindenberger, Pavel Kalvoda, Marek Konieczny, Onur Mutlu, Torsten Hoefler



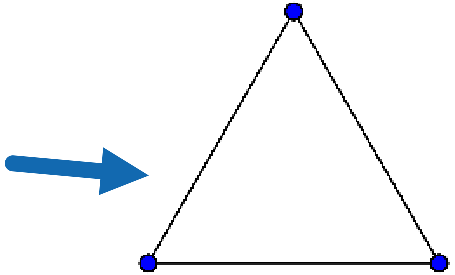
Graph Mining



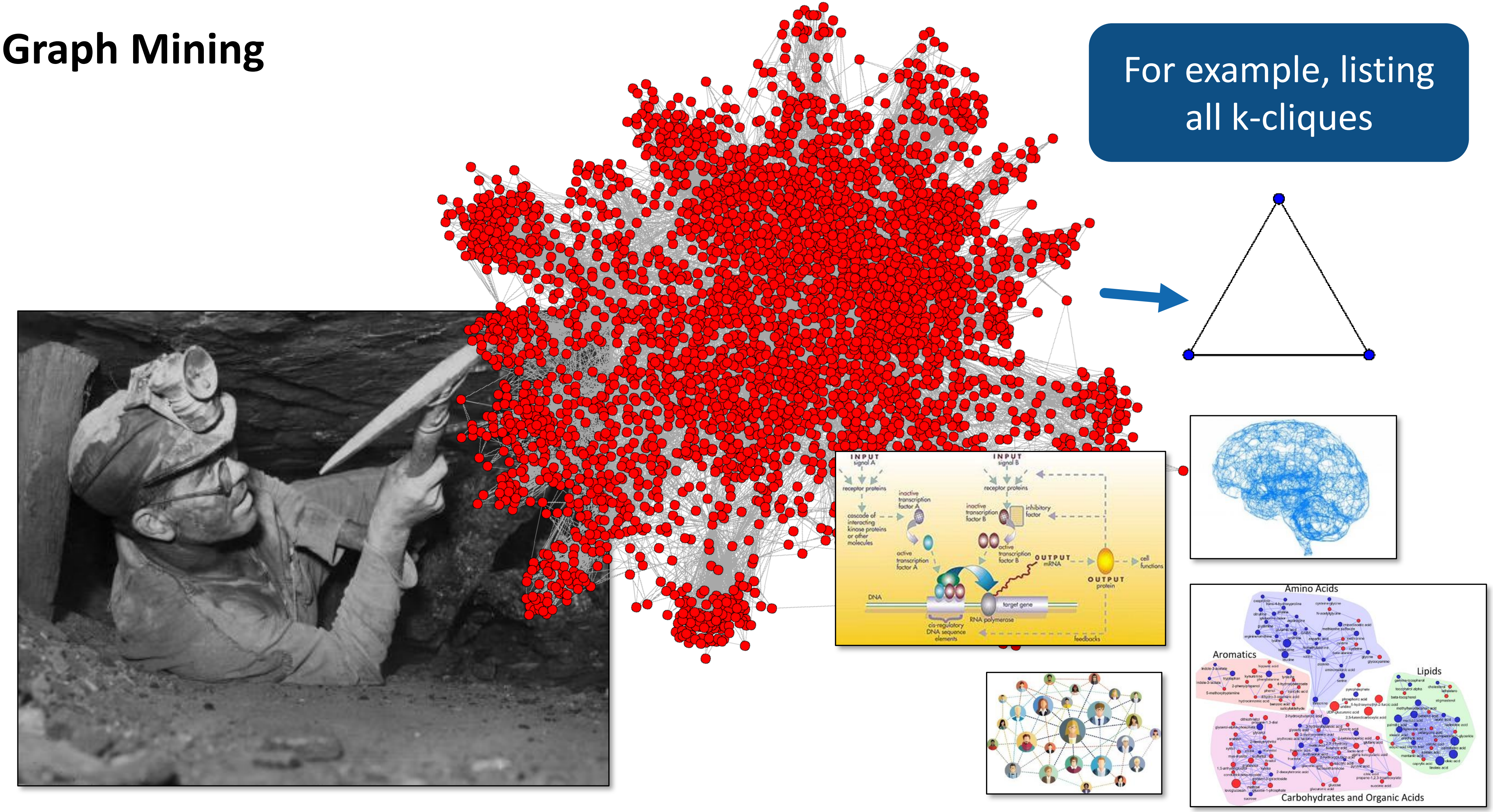
Graph Mining



For example, listing all k-cliques



Graph Mining

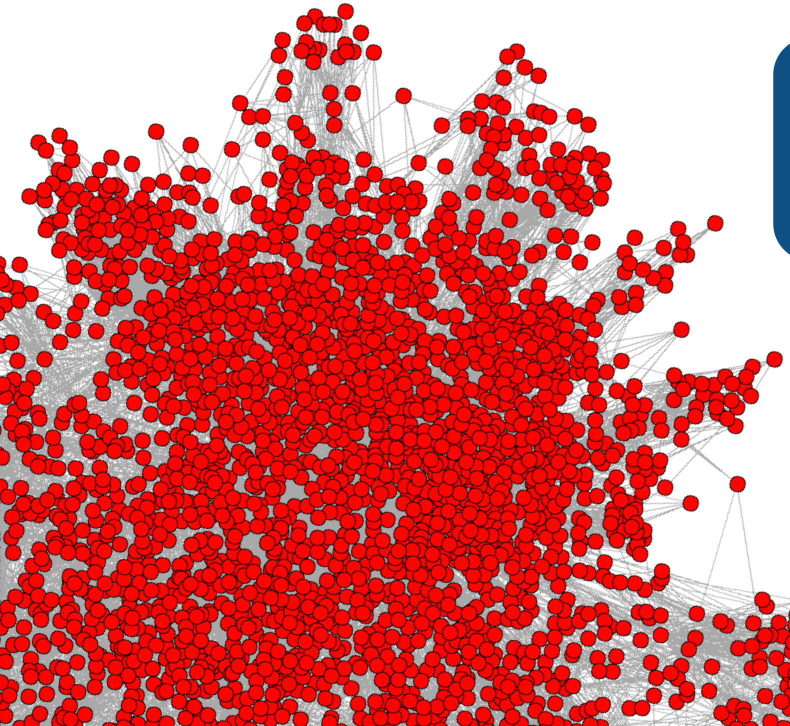

INPUT
signal A
INPUT
signal B
receptor proteins
inactive transcription factor A
inactive transcription factor B
inhibitory factor
cascade of interacting kinase proteins or other molecules
active transcription factor A
active transcription factor B
OUTPUT
mRNA
OUTPUT
protein
cell functions
DNA
target gene
RNA polymerase
cis-regulatory DNA sequence elements
feedbacks
Amino Acids
Aromatics
Lipids
Carbohydrates and Organic Acids

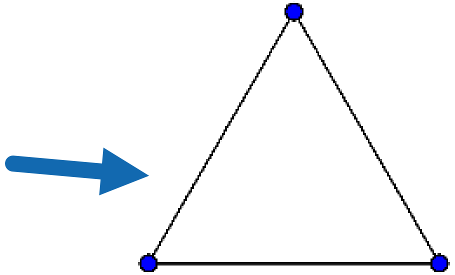
For example, listing all k-cliques

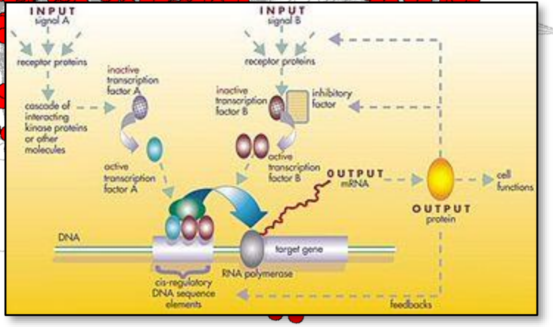
Graph Mining

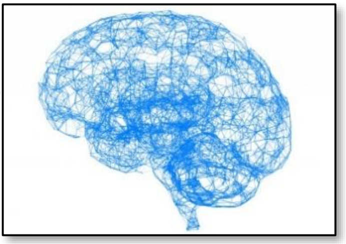
Challenges?

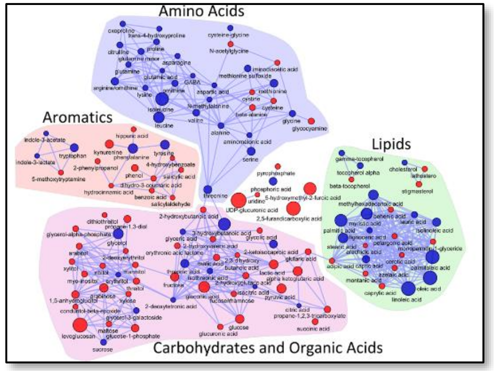
For example, listing all k-cliques






INPUT
signal A
INPUT
signal B
receptor proteins
receptor proteins
inactive transcription factor A
inactive transcription factor B
inhibitory factor
cascade of interacting kinase proteins or other molecules
active transcription factor A
active transcription factor B
OUTPUT
mRNA
cell functions
OUTPUT
protein
DNA
target gene
RNA polymerase
cis-regulatory DNA sequence elements
feedbacks

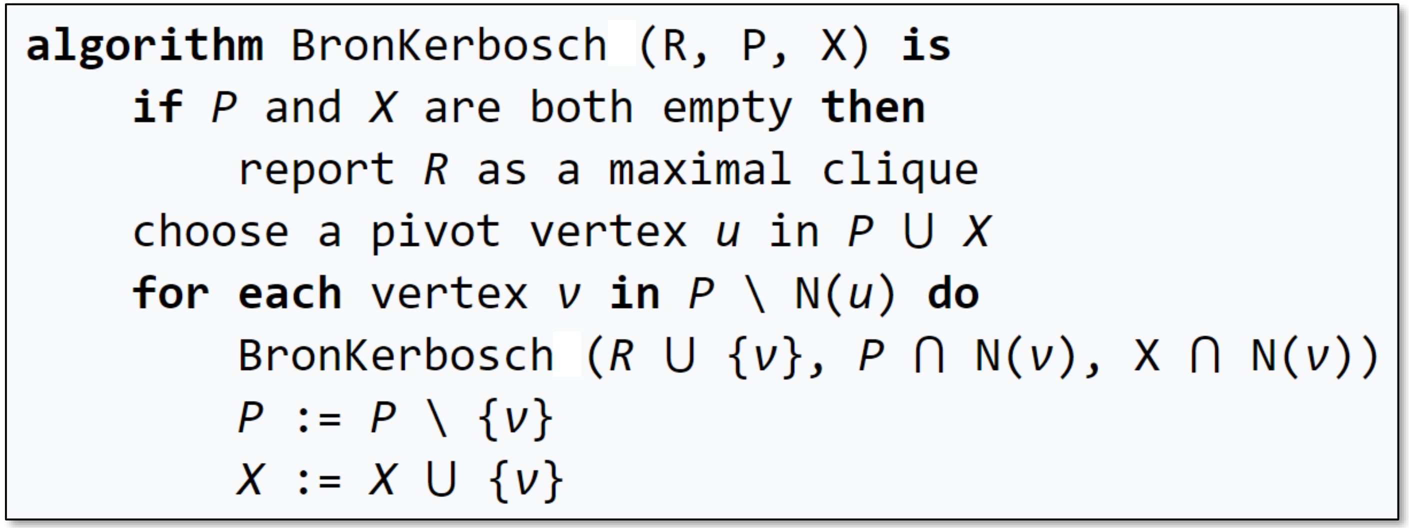




Graph Mining: Challenges

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

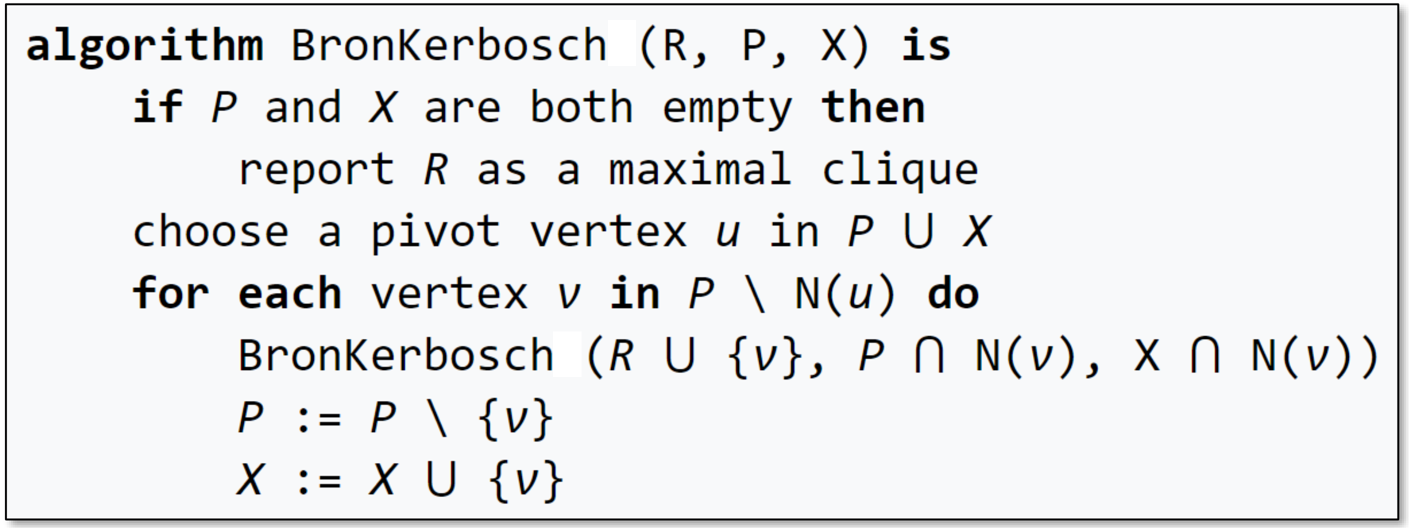


```
algorithm BronKerbosch (R, P, X) is
    if $P$ and $X$ are both empty then
        report $R$ as a maximal clique
    choose a pivot vertex $u$ in $P \cup X$
    for each vertex $v$ in $P \setminus N(u)$ do
        BronKerbosch ($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
        $P := P \setminus \{v\}$
        $X := X \cup \{v\}$
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

Complex algorithm structure, deeply recursive, no notion of iterations

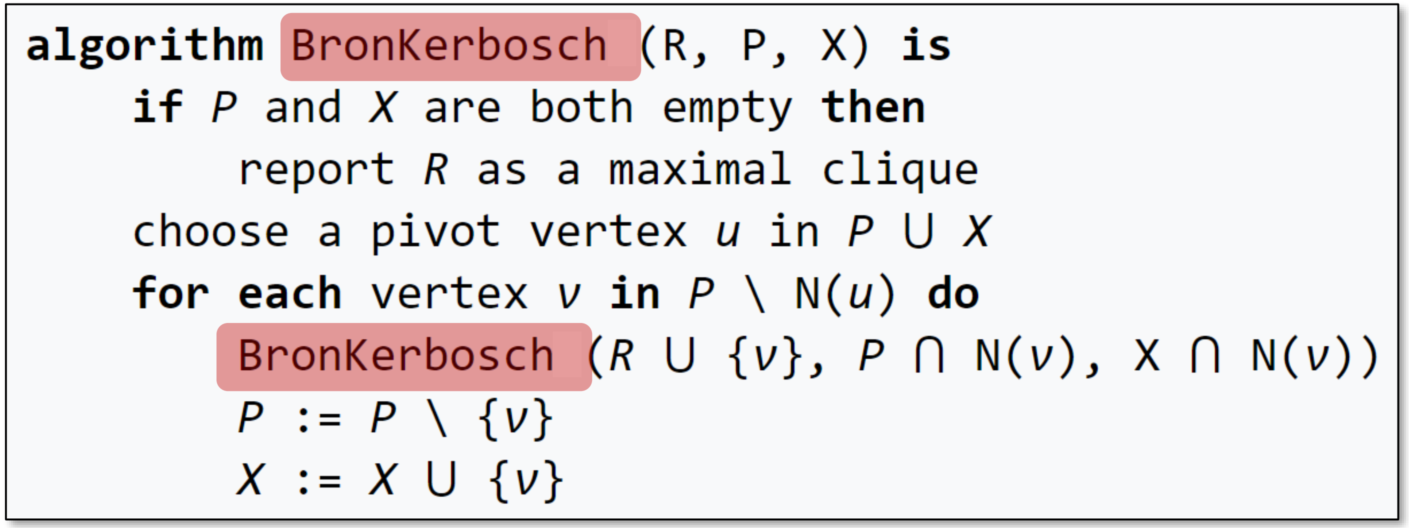


```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

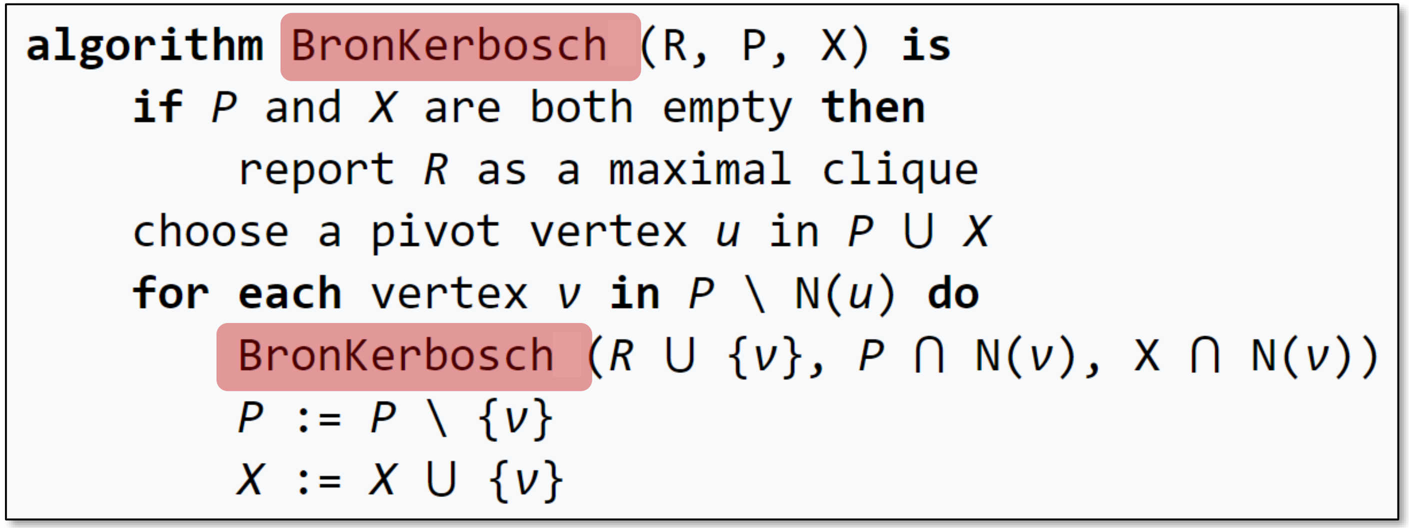
Complex algorithm structure, deeply recursive, no notion of iterations



```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



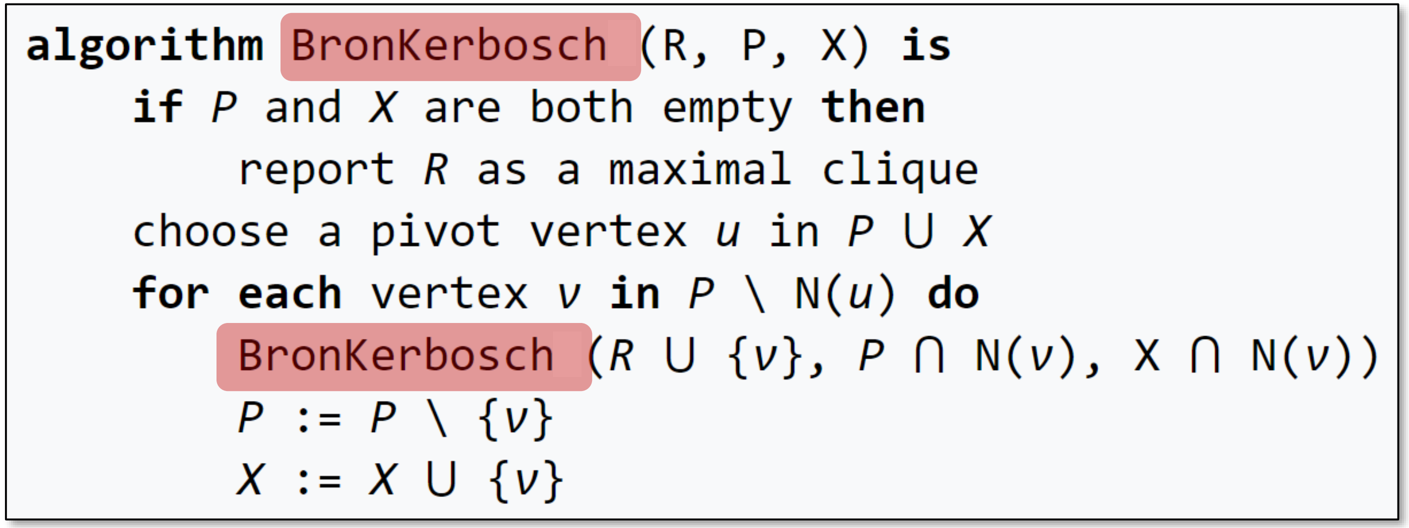
```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



```
algorithm BronKerbosch(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

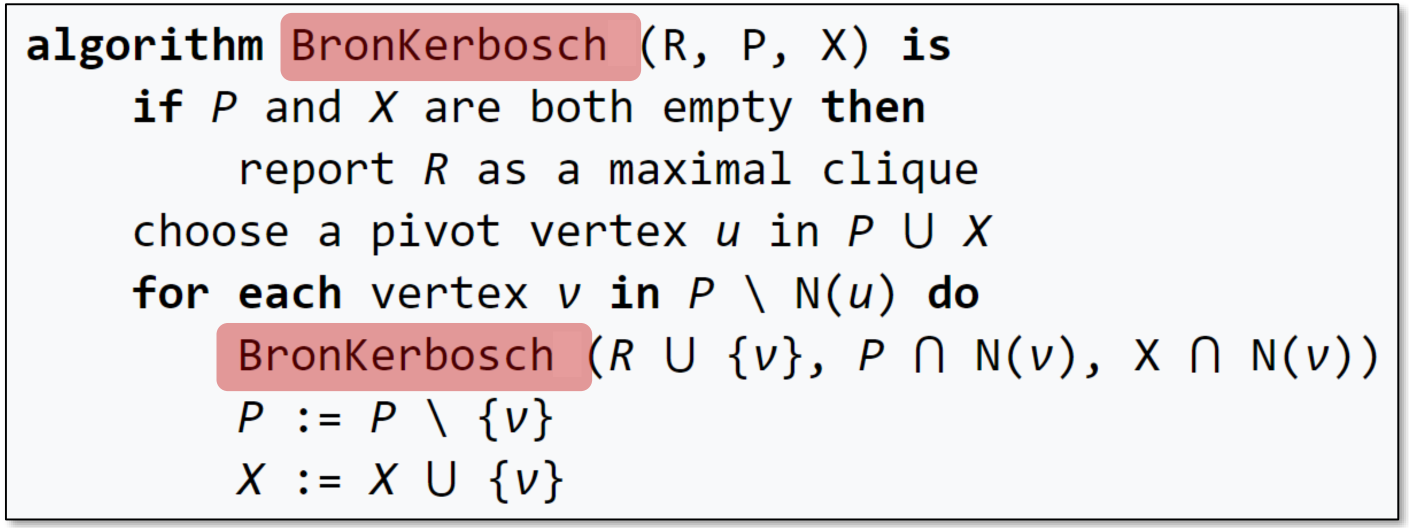
```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

...Repeat several times

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

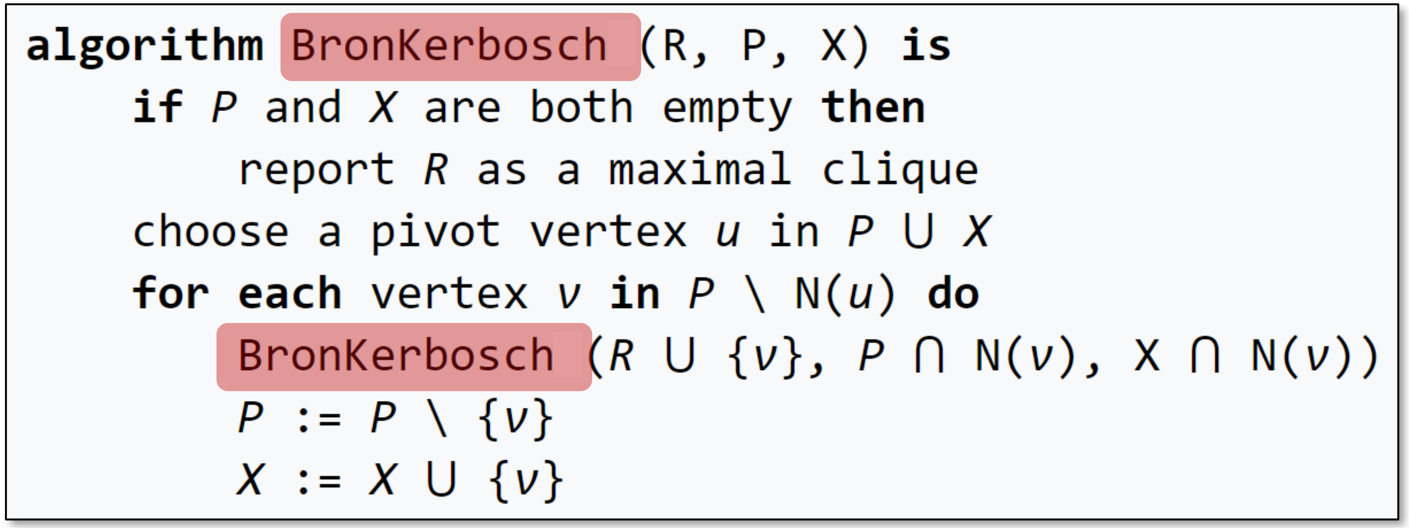
```
while not done
  for all vertices v:
    send updates over outgoing edges of v
  for all vertices v:
    apply updates from inbound edges of v
```

...Repeat several times

Not very complicated

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



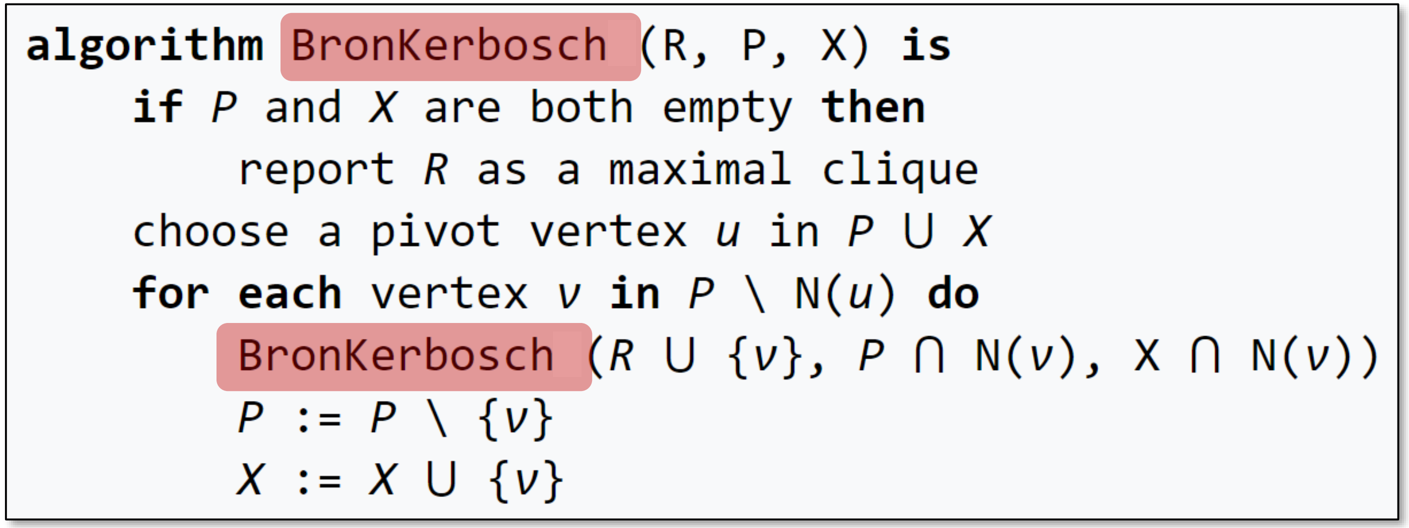
```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing



```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

Complex algorithm structure, deeply recursive, no notion of iterations

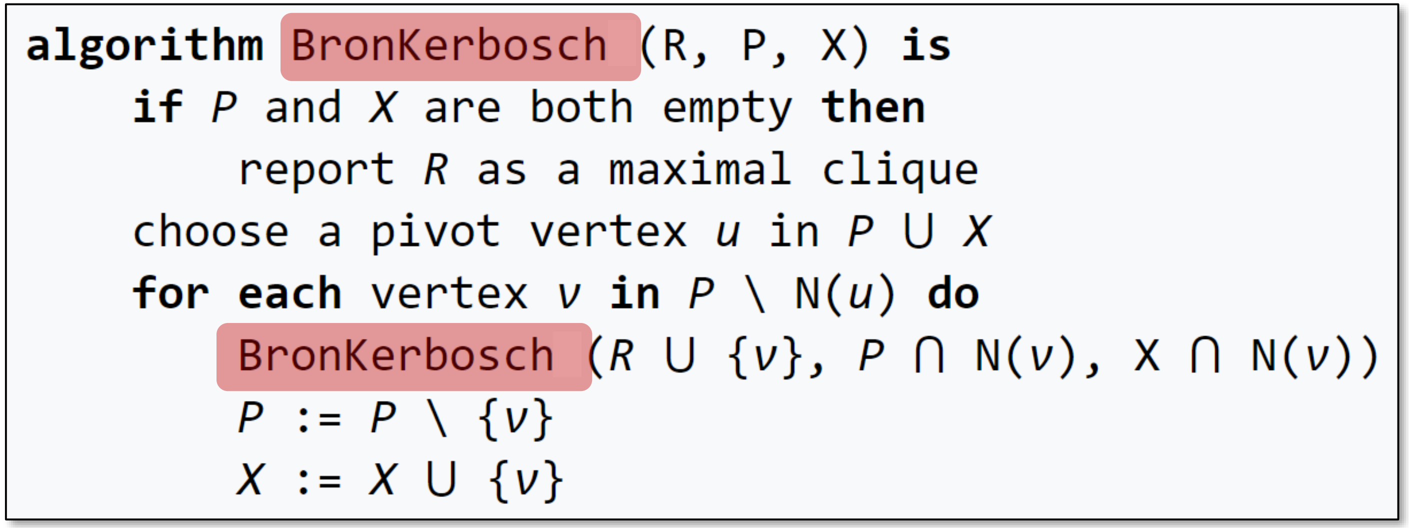
Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

Many other algorithms with similar properties



```
algorithm BronKerbosch(R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch(R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

Many other algorithms with similar properties

k-clique listing

Clustering

Subgraph isomorphism

Vertex orderings

Dense subgraph discovery

Link prediction

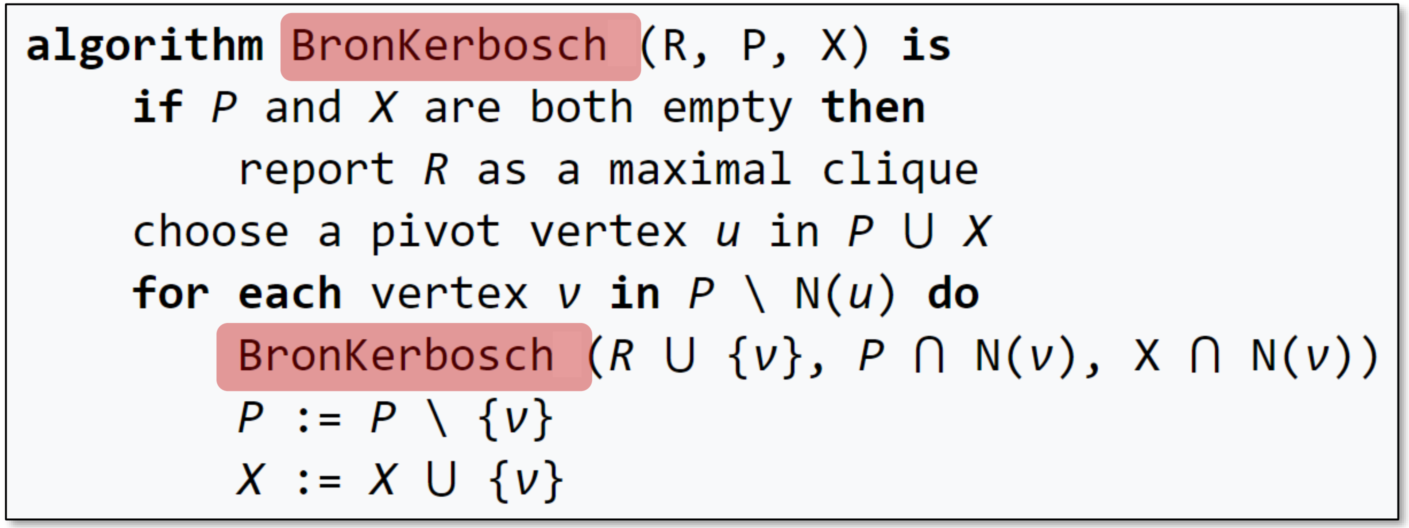
Frequent subgraph mining

Vertex similarity

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME



```
algorithm BronKerbosch(R, P, X) is
    if $P$ and $X$ are both empty then
        report $R$ as a maximal clique
    choose a pivot vertex $u$ in $P \cup X$
    for each vertex $v$ in $P \setminus N(u)$ do
        BronKerbosch($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
        $P := P \setminus \{v\}$
        $X := X \cup \{v\}$
```

Graph Mining: Challenges

Example: the Bron-Kerbosch algorithm for maximal clique listing

Many other algorithms with similar properties

k-clique listing

Clustering

Dense subgraph discovery

Subgraph isomorphism

Vertex orderings

Vertex similarity

Frequent subgraph mining

Link prediction

Complex algorithm structure, deeply recursive, no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Goal: construct a high-performance algorithm solving a selected graph mining problem

How to achieve this goal?

Gr

Ex
Bro
alg
maximal clique listing

with similar properties

lustering

Vertex
rderings

Link prediction

Frequent subgraph mining

discovery

Vertex similarity

hm
cursive,
no notion of iterations

Non-straightforward parallelism, complicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Gr

Goal: construct a high-performance algorithm solving a selected graph mining problem

Ex
Bro
alg
maximal clique listing

with similar properties

Link prediction

lustering

Vertex
derings

Frequent subgraph mining

discovery

How to achieve this goal?

hm
cursive,
no notion of iterations

One has to address several issues...

rward
plicated
memory access patterns

Many algorithms are NP-complete or even EXPTIME

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Goal: construct a high-performance algorithm solving a selected graph mining problem

How to achieve this goal?

One has to address several issues...

What are relevant mining baselines and datasets?

Gr

Ex
Bro
alg
maximal
...e listing

with similar properties

Link prediction

...lustering

...Vertex ...rderings

Frequent subgraph mining

...discovery

...hm ...cursive, no notion of iterations

...rward ...plicated memory access patterns

Many algorithms are NP-complete or even EXPTIME

```
alg
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch(R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Goal: construct a high-performance algorithm solving a selected graph mining problem

?

How to achieve this goal?

One has to address several issues...

?

What are relevant mining baselines and datasets?

?

How to effectively develop new efficient baselines?

Gr

Ex

Bro

alg

maximal

with similar properties

lustering

Vertex

derings

Frequent subgraph mining

Link prediction

discovery

m

cursive,

no notion of iterations

rward

plicated

memory access patterns

Many algorithms are NP-complete or even EXPTIME

```
alg
    report R as a maximal clique
                                  ), X ∩ N(v))
    X := X U {v}
```

Gr

Goal: construct a high-performance algorithm solving a selected graph mining problem

lustering

How to achieve this goal?

hm
cursive,
no notion of iterations

Ex
Bro
alg
maximal
o listing

with similar
properties

Link prediction

Vertex
derings

Frequent
subgraph
mining

discovery

One has to address several issues...

rward
plicated
memory access patterns

What are relevant mining baselines and datasets?

alg

```
report R as a maximal clique
), X ∩ N(v))
X := X ∪ {v}
```

How to effectively develop new efficient baselines?

How to analyze performance/others, using what metrics?

Goal: construct a high-performance algorithm solving a selected graph mining problem

How to achieve this goal?

One has to address several issues...

What are relevant mining baselines and datasets?

How to effectively develop new efficient baselines?

How to analyze performance/others, using what metrics?

...

Gr

Ex
Br
alg
maximal

with similar
properties

lustering

Vertex
derings

Link prediction

Frequent
subgraph
mining

discovery

hm
cursive,
no notion of iterations

rward
plicated
memory access patterns

```
alg
      report R as a maximal clique
                                 ), X ∩ N(v))
      X := X U {v}
```

Goal: construct a high-performance algorithm solving a selected graph mining problem

How to achieve this goal?

One has to address several issues...

What are relevant mining baselines and datasets?

How to effectively develop new efficient baselines?

How to analyze performance/others, using what metrics?

...

ustering

Vertex
derings

Frequent
subgraph
mining

Link prediction

maximal

with similar
properties

cursive,
no notion of iterations

rward
plicated
memory access patterns

alg

report R as a maximal clique

), X ∩ N(v))

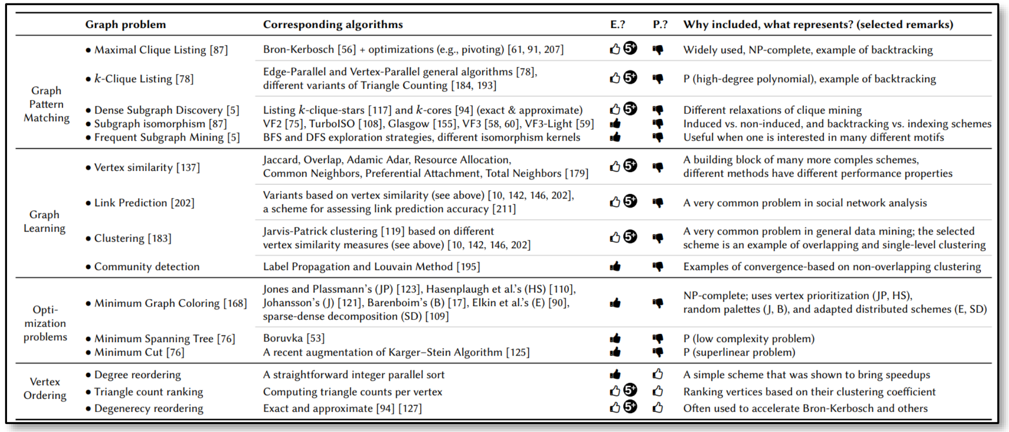
$X := X \cup \{v\}$

GraphMineSuite

GraphMineSuite (GMS) comes with...

GraphMineSuite (GMS) comes with...

1 ... **Benchmark specification** prescribing representative *problems*, *algorithms*, and *datasets*

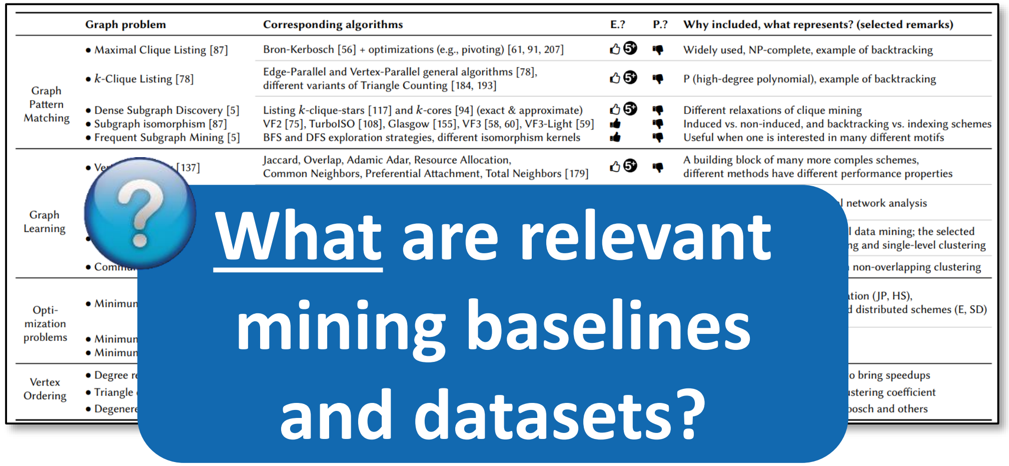


	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	🖒❺+	👎	Widely used, NP-complete, example of backtracking
	• k-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	🖒❺+	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing k-clique-stars [117] and k-cores [94] (exact & approximate)	🖒❺+	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Vertex similarity [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	🖒❺+	👎	A building block of many more comples schemes, different methods have different performance properties
	• Link Prediction [202]	Variants based on vertex similarity (see above) [10, 142, 146, 202], a scheme for assessing link prediction accuracy [211]	🖒❺+	👎	A very common problem in social network analysis
	• Clustering [183]	Jarvis-Patrick clustering [119] based on different vertex similarity measures (see above) [10, 142, 146, 202]	🖒❺+	👎	A very common problem in general data mining; the selected scheme is an example of overlapping and single-level clustering
	• Community detection	Label Propagation and Louvain Method [195]	👍	👎	Examples of convergence-based on non-overlapping clustering
Opti-mization problems	• Minimum Graph Coloring [168]	Jones and Plassmann's (JP) [123], Hasenplaugh et al.'s (HS) [110], Johansson's (J) [121], Barenboim's (B) [17], Elkin et al.'s (E) [90], sparse-dense decomposition (SD) [109]	👍	👎	NP-complete; uses vertex prioritization (JP, HS), random palettes (J, B), and adapted distributed schemes (E, SD)
	• Minimum Spanning Tree [76]	Boruvka [53]	👍	👎	P (low complexity problem)
	• Minimum Cut [76]	A recent augmentation of Karger–Stein Algorithm [125]	👍	👎	P (superlinear problem)
Vertex Ordering	• Degree reordering	A straightforward integer parallel sort	👍	🖒	A simple scheme that was shown to bring speedups
	• Triangle count ranking	Computing triangle counts per vertex	🖒❺+	🖒	Ranking vertices based on their clustering coefficient
	• Degenerecy reordering	Exact and approximate [94] [127]	🖒❺+	🖒	Often used to accelerate Bron-Kerbosch and others

GraphMineSuite (GMS) comes with...

1

... **Benchmark specification** prescribing representative *problems*, *algorithms*, and *datasets*



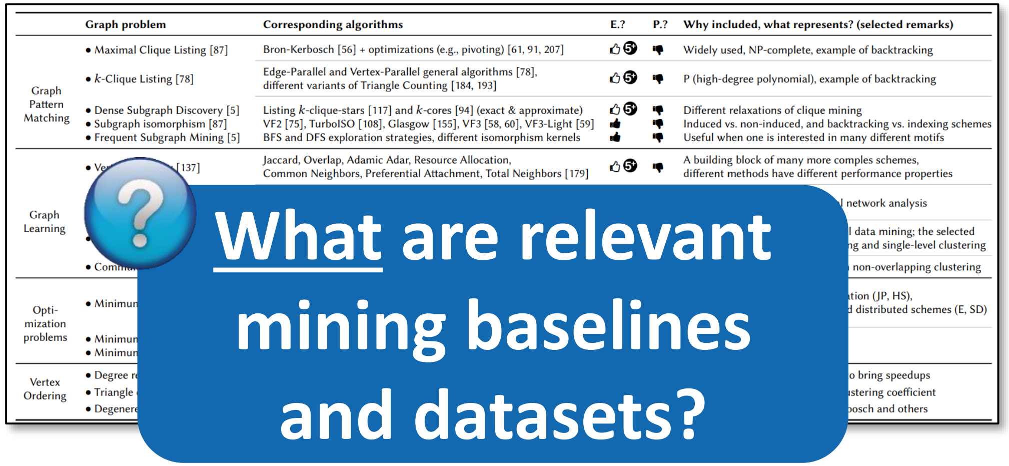
	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	👍 5+	👎	Widely used, NP-complete, example of backtracking
	• *k*-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	👍 5+	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing *k*-clique-stars [117] and *k*-cores [94] (exact & approximate)	👍 5+	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Ve [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	👍 5+	👎	A building block of many more comples schemes, different methods have different performance properties
					l network analysis
	•				l data mining; the selected ng and single-level clustering
	• Comm				non-overlapping clustering
Opti-mization problems	• Minimun				tion (JP, HS), d distributed schemes (E, SD)
	• Minimun • Minimun				
Vertex Ordering	• Degree r • Triangle • Degenere				o bring speedups stering coefficient bosch and others

What are relevant mining baselines and datasets?

GraphMineSuite (GMS) comes with...

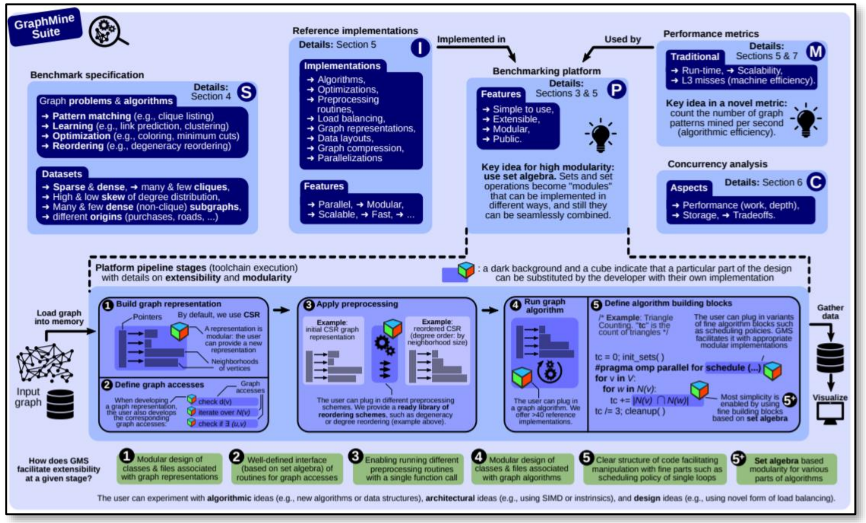
1 ... **Benchmark specification** prescribing representative *problems*, *algorithms*, and *datasets*

2 ... Software **platform** with **reference implementations** based on **set algebraic formulations** for *programmability* & *high performance*



	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	👍 5+	👎	Widely used, NP-complete, example of backtracking
	• *k*-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	👍 5+	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing *k*-clique-stars [117] and *k*-cores [94] (exact & approximate)	👍 5+	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Ve [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	👍 5+	👎	A building block of many more comples schemes, different methods have different performance properties
					l network analysis
	•				l data mining; the selected ng and single-level clustering
	• Comm				non-overlapping clustering
Opti-mization problems	• Minimun				tion (JP, HS), d distributed schemes (E, SD)
	• Minimun • Minimun				
Vertex Ordering	• Degree r				o bring speedups
	• Triangle				stering coefficient
	• Degenere				osch and others

What are relevant mining baselines and datasets?


GraphMine Suite
Reference implementations
Details: Section 5
I
Implementations
→ Algorithms,
→ Optimizations,
→ Preprocessing routines,
→ Load balancing,
→ Graph representations,
→ Data layouts,
→ Graph compression,
→ Parallelizations
Features
→ Parallel, → Modular,
→ Scalable, → Fast, → ...
Implemented in
Used by
Benchmark specification
Details: Section 4
S
Graph problems & algorithms
→ Pattern matching (e.g., clique listing)
→ Learning (e.g., link prediction, clustering)
→ Optimization (e.g., coloring, minimum cuts)
→ Reordering (e.g., degeneracy reordering)
Datasets
→ Sparse & dense, → many & few cliques,
→ High & low skew of degree distribution,
→ Many & few dense (non-clique) subgraphs,
→ different origins (purchases, roads, ...)
Benchmarking platform
Details: Sections 3 & 5
P
Features
→ Simple to use,
→ Extensible,
→ Modular,
→ Public.
Performance metrics
Details: Sections 5 & 7
M
Traditional
→ Run-time, → Scalability,
→ L3 misses (machine efficiency).
Concurrency analysis
Details: Section 6
C
Aspects
→ Performance (work, depth),
→ Storage, → Tradeoffs.
Platform pipeline stages (toolchain execution)
with details on extensibility and modularity
Load graph into memory
Input graph
1 Build graph representation
Pointers
By default, we use CSR
Neighborhoods of vertices
2 Define graph accesses
Graph accesses
check d(v)
iterate over N(v)
check if ∃ (u,v)
3 Apply preprocessing
Example: initial CSR graph representation
Example: reordered CSR (degree order: by neighborhood size)
4 Run graph algorithm
5 Define algorithm building blocks
5+
Gather data
Visualize
How does GMS facilitate extensibility at a given stage?
1
2
3
4
5
5+

Key idea for high modularity: use set algebra. Sets and set operations become "modules" that can be implemented in different ways, and still they can be seamlessly combined.

Key idea in a novel metric: count the number of graph patterns mined per second (algorithmic efficiency).

: a dark background and a cube indicate that a particular part of the design can be substituted by the developer with their own implementation

A representation is modular: the user can provide a new representation

When developing a graph representation, the user also develops the corresponding graph accesses:

The user can plug in different preprocessing schemes. We provide a ready library of reordering schemes, such as degeneracy or degree reordering (example above).

The user can plug in a graph algorithm. We offer >40 reference implementations.

```
/* Example: Triangle Counting. "tc" is the count of triangles */
tc = 0; init_sets( )
#pragma omp parallel for schedule (...)
for v in V:
  for w in N(v):
    tc += |N(v) ∩ N(w)|
tc /= 3; cleanup( )
```

The user can plug in variants of fine algorithm blocks such as scheduling policies. GMS facilitates it with appropriate modular implementations

Most simplicity is enabled by using fine building blocks based on set algebra

1 Modular design of classes & files associated with graph representations

2 Well-defined interface (based on set algebra) of routines for graph accesses

3 Enabling running different preprocessing routines with a single function call

4 Modular design of classes & files associated with graph algorithms

5 Clear structure of code facilitating manipulation with fine parts such as scheduling policy of single loops

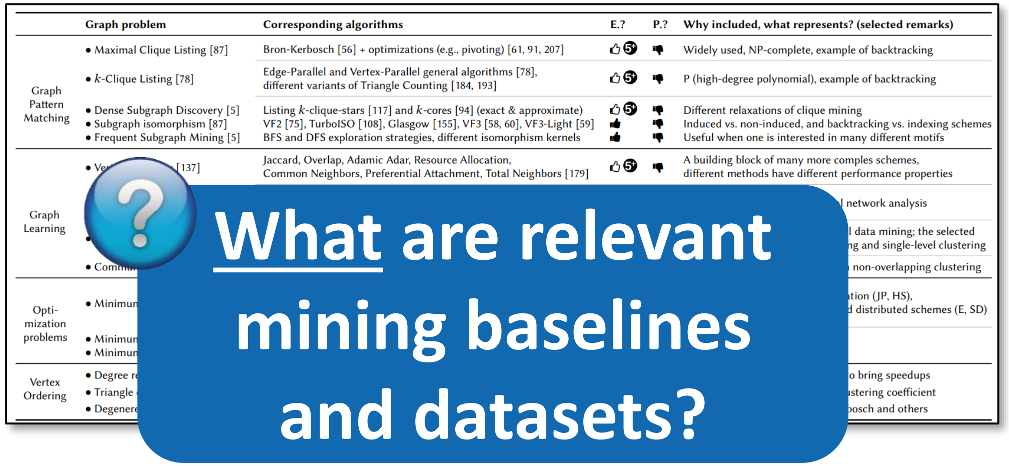
5+ Set algebra based modularity for various parts of algorithms

The user can experiment with algorithmic ideas (e.g., new algorithms or data structures), architectural ideas (e.g., using SIMD or instrinsics), and design ideas (e.g., using novel form of load balancing).

GraphMineSuite (GMS) comes with...

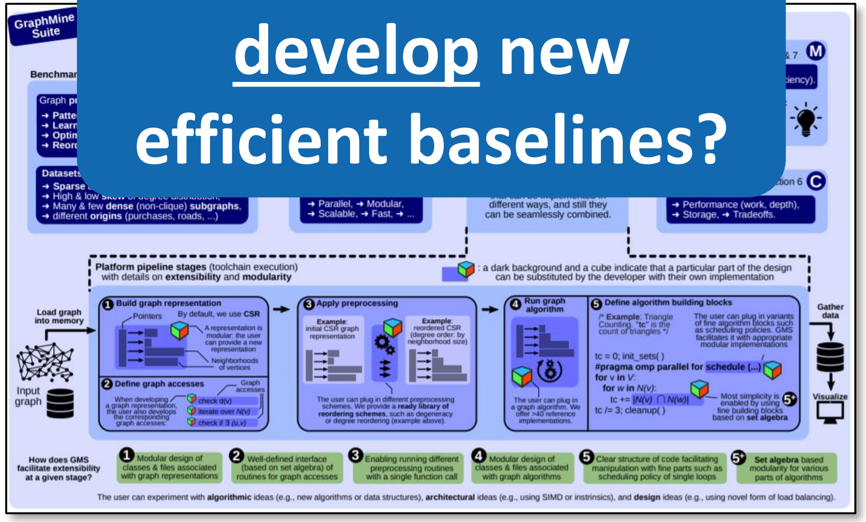
1 ... **Benchmark specification** prescribing representative *problems*, *algorithms*, and *datasets*

2 ... Software **platform** with **reference implementations** based on **set algebraic formulations** for *programmability* & *high performance*



	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	👍 5+	👎	Widely used, NP-complete, example of backtracking
	• *k*-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	👍 5+	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing *k*-clique-stars [117] and *k*-cores [94] (exact & approximate)	👍 5+	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Ve [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	👍 5+	👎	A building block of many more comples schemes, different methods have different performance properties
					l network analysis
	•				l data mining; the selected ng and single-level clustering
	• Comm				non-overlapping clustering
Opti-mization problems	• Minimun				tion (JP, HS), d distributed schemes (E, SD)
	• Minimun • Minimun				
Vertex Ordering	• Degree r				o bring speedups
	• Triangle				stering coefficient
	• Degenere				osch and others

What are relevant mining baselines and datasets?


GraphMine Suite
Benchmar
Graph pr
→ Patte
→ Learn
→ Optin
→ Reord
Datasets
→ Sparse
→ High & low skew of degree distribution,
→ Many & few dense (non-clique) subgraphs,
→ different origins (purchases, roads, ...)
→ Parallel, → Modular,
→ Scalable, → Fast, → ...
different ways, and still they can be seamlessly combined.
tion 6 C
→ Performance (work, depth),
→ Storage, → Tradeoffs.
& 7 M
iency).
Platform pipeline stages (toolchain execution) with details on extensibility and modularity
: a dark background and a cube indicate that a particular part of the design can be substituted by the developer with their own implementation
Load graph into memory
Input graph
1 Build graph representation
Pointers
By default, we use CSR
A representation is modular: the user can provide a new representation
Neighborhoods of vertices
2 Define graph accesses
Graph accesses
When developing a graph representation, the user also develops the corresponding graph accesses:
check d(v)
iterate over N(v)
check if ∃ (u,v)
3 Apply preprocessing
Example: initial CSR graph representation
Example: reordered CSR (degree order: by neighborhood size)
The user can plug in different preprocessing schemes. We provide a ready library of reordering schemes, such as degeneracy or degree reordering (example above).
4 Run graph algorithm
The user can plug in a graph algorithm. We offer >40 reference implementations.
5 Define algorithm building blocks
The user can plug in variants of fine algorithm blocks such as scheduling policies. GMS facilitates it with appropriate modular implementations
Most simplicity is enabled by using fine building blocks based on set algebra 5+
Gather data
Visualize
How does GMS facilitate extensibility at a given stage?
1 Modular design of classes & files associated with graph representations
2 Well-defined interface (based on set algebra) of routines for graph accesses
3 Enabling running different preprocessing routines with a single function call
4 Modular design of classes & files associated with graph algorithms
5 Clear structure of code facilitating manipulation with fine parts such as scheduling policy of single loops
5+ Set algebra based modularity for various parts of algorithms

```
/* Example: Triangle Counting. "tc" is the count of triangles */
tc = 0; init_sets( )
#pragma omp parallel for schedule (...)
for v in V:
  for w in N(v):
    tc += |N(v) ∩ N(w)|
tc /= 3; cleanup( )
```

The user can experiment with **algorithmic** ideas (e.g., new algorithms or data structures), **architectural** ideas (e.g., using SIMD or instrinsics), and **design** ideas (e.g., using novel form of load balancing).

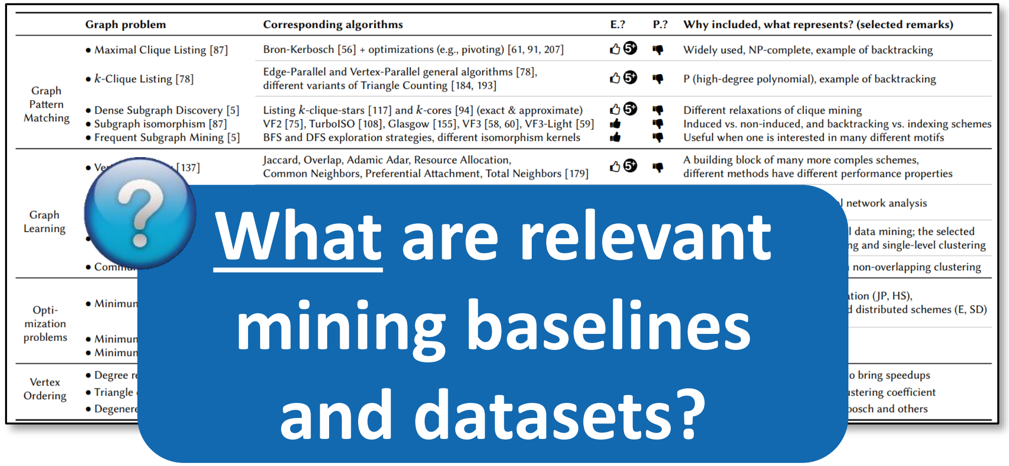
How to effectively **develop** new efficient baselines?

GraphMineSuite (GMS) comes with...

1 ... **Benchmark specification** prescribing representative *problems, algorithms*, and *datasets*

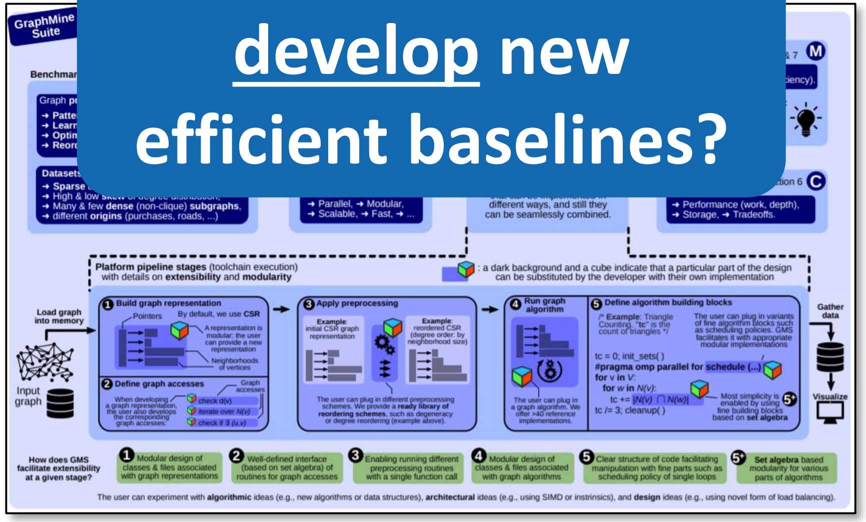
2 ... Software **platform** with **reference implementations** based on **set algebraic formulations** for *programmability* & *high performance*

3 ... **Performance metrics, e.g.,** to assesses *algorithmic throughput*



	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	👍 5⁺	👎	Widely used, NP-complete, example of backtracking
	• k-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	👍 5⁺	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing k-clique-stars [117] and k-cores [94] (exact & approximate)	👍 5⁺	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Ve [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	👍 5⁺	👎	A building block of many more comples schemes, different methods have different performance properties
					l network analysis
	•				l data mining; the selected ng and single-level clustering
	• Comm				non-overlapping clustering
Opti-mization problems	• Minimun				tion (JP, HS), d distributed schemes (E, SD)
	• Minimun • Minimun				
Vertex Ordering	• Degree r				o bring speedups
	• Triangle				stering coefficient
	• Degenere				bosch and others

What are relevant mining baselines and datasets?


GraphMine Suite
Benchmar
Graph pr
➜ Patte
➜ Learn
➜ Optin
➜ Reor
Datasets
➜ Sparse
➜ High & low
➜ Many & few dense (non-clique) subgraphs,
➜ different origins (purchases, roads, ...)
➜ Parallel, ➜ Modular,
➜ Scalable, ➜ Fast, ➜ ...
& 7
M
iency).
tion 6
C
➜ Performance (work, depth),
➜ Storage, ➜ Tradeoffs.
Load graph into memory
Input graph
1 Build graph representation
Pointers
By default, we use CSR
Neighborhoods of vertices
2 Define graph accesses
Graph accesses
check d(v)
iterate over N(v)
check if ∃ (u,v)
3 Apply preprocessing
Example: initial CSR graph representation
Example: reordered CSR (degree order: by neighborhood size)
4 Run graph algorithm
5 Define algorithm building blocks
Gather data
Visualize
How does GMS facilitate extensibility at a given stage?

different ways, and still they can be seamlessly combined.

Platform pipeline stages (toolchain execution) with details on extensibility and modularity

: a dark background and a cube indicate that a particular part of the design can be substituted by the developer with their own implementation

A representation is modular: the user can provide a new representation

When developing a graph representation, the user also develops the corresponding graph accesses:

The user can plug in different preprocessing schemes. We provide a **ready library of reordering schemes**, such as degeneracy or degree reordering (example above).

The user can plug in a graph algorithm. We offer >40 reference implementations.

The user can plug in variants of fine algorithm blocks such as scheduling policies. GMS facilitates it with appropriate modular implementations

```
/* Example: Triangle Counting. "tc" is the count of triangles */
tc = 0; init_sets( )
#pragma omp parallel for schedule (...)
for v in V:
  for w in N(v):
    tc += |N(v) ∩ N(w)|
tc /= 3; cleanup( )
```

Most simplicity is enabled by using fine building blocks based on **set algebra** 5⁺

1 Modular design of classes & files associated with graph representations

2 Well-defined interface (based on set algebra) of routines for graph accesses

3 Enabling running different preprocessing routines with a single function call

4 Modular design of classes & files associated with graph algorithms

5 Clear structure of code facilitating manipulation with fine parts such as scheduling policy of single loops

5⁺ **Set algebra** based modularity for various parts of algorithms

The user can experiment with **algorithmic** ideas (e.g., new algorithms or data structures), **architectural** ideas (e.g., using SIMD or instrinsics), and **design** ideas (e.g., using novel form of load balancing).

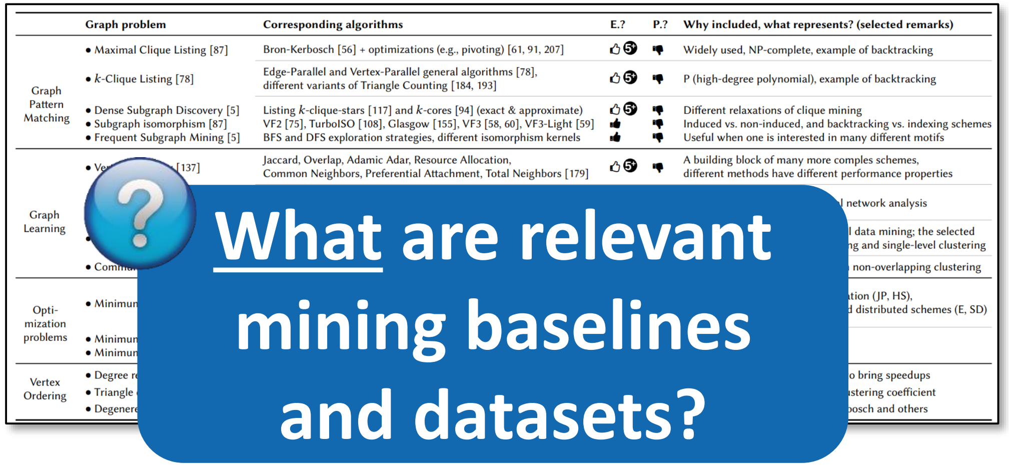
How to effectively **develop** new efficient baselines?

GraphMineSuite (GMS) comes with...

1 ... **Benchmark specification** prescribing representative *problems, algorithms*, and *datasets*

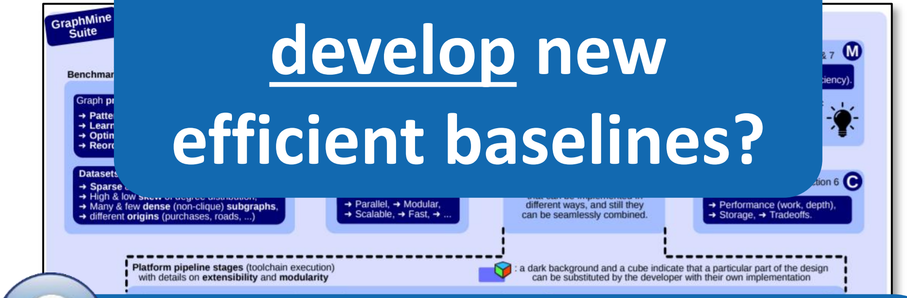
2 ... Software **platform** with **reference implementations** based on **set algebraic formulations** for *programmability* & *high performance*

3 ... **Performance metrics, e.g.,** to assesses *algorithmic throughput*



	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	👍 5+	👎	Widely used, NP-complete, example of backtracking
	• *k*-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	👍 5+	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing *k*-clique-stars [117] and *k*-cores [94] (exact & approximate)	👍 5+	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Ve [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	👍 5+	👎	A building block of many more comples schemes, different methods have different performance properties
					l network analysis
					l data mining; the selected ng and single-level clustering
	• Comm				non-overlapping clustering
Opti-mization problems	• Minimun				tion (JP, HS), d distributed schemes (E, SD)
	• Minimun				
	• Minimun				
Vertex Ordering	• Degree r				o bring speedups
	• Triangle				stering coefficient
	• Degenere				bosch and others

What are relevant mining baselines and datasets?



GraphMine Suite

Benchmar

Graph pr
→ Patte
→ Learn
→ Optin
→ Reord

Datasets
→ Sparse
→ High & low skew of degree distribution,
→ Many & few **dense** (non-clique) **subgraphs**,
→ different **origins** (purchases, roads, ...)

→ Parallel, → Modular,
→ Scalable, → Fast, → ...

different ways, and still they can be seamlessly combined.

& 7 M

iency).

tion 6 C

→ Performance (work, depth),
→ Storage, → Tradeoffs.

Platform pipeline stages (toolchain execution) with details on **extensibility** and **modularity**

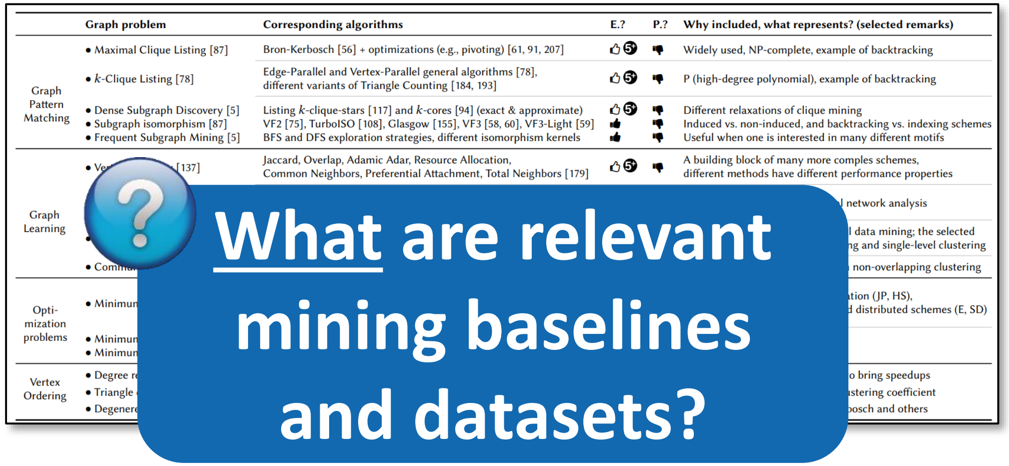
: a dark background and a cube indicate that a particular part of the design can be substituted by the developer with their own implementation

How to effectively **develop** new efficient baselines?

How to **analyze** performance/others, using **what** metrics?

GraphMineSuite (GMS) comes with...

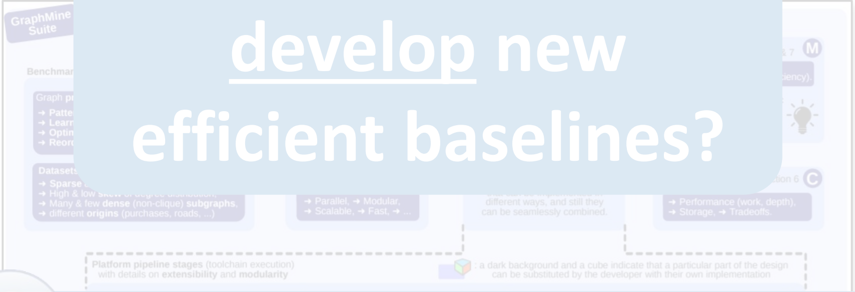
1 ... **Benchmark specification** prescribing representative *problems*, *algorithms*, and *datasets*



	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	👍 5+	👎	Widely used, NP-complete, example of backtracking
	• *k*-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	👍 5+	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing *k*-clique-stars [117] and *k*-cores [94] (exact & approximate)	👍 5+	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Ve [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	👍 5+	👎	A building block of many more comples schemes, different methods have different performance properties
					l network analysis
	•				l data mining; the selected ng and single-level clustering
	• Comm				non-overlapping clustering
Opti-mization problems	• Minimun				tion (JP, HS), d distributed schemes (E, SD)
	• Minimun				
	• Minimun				
Vertex Ordering	• Degree r				o bring speedups
	• Triangle				stering coefficient
	• Degenere				osch and others

? **What** are relevant mining baselines and datasets?

2 ... Software **platform** with **reference implementations** based on **set algebraic formulations** for *programmability* & *high performance*


GraphMine Suite
M
Benchman
iency).
Graph pr
→ Patte
→ Learn
→ Optin
→ Reord
Datasets
→ Sparse
→ High & low skew of degree distribution,
→ Many & few dense (non-clique) subgraphs,
→ different origins (purchases, roads, ...)
→ Parallel, → Modular,
→ Scalable, → Fast, → ...
different ways, and still they
can be seamlessly combined.
ction 6
C
→ Performance (work, depth),
→ Storage, → Tradeoffs.
Platform pipeline stages (toolchain execution)
with details on extensibility and modularity
: a dark background and a cube indicate that a particular part of the design
can be substituted by the developer with their own implementation

? **How** to effectively **develop** new efficient baselines?

3 ... **Performance metrics, e.g.,** to assesses *algorithmic throughput*

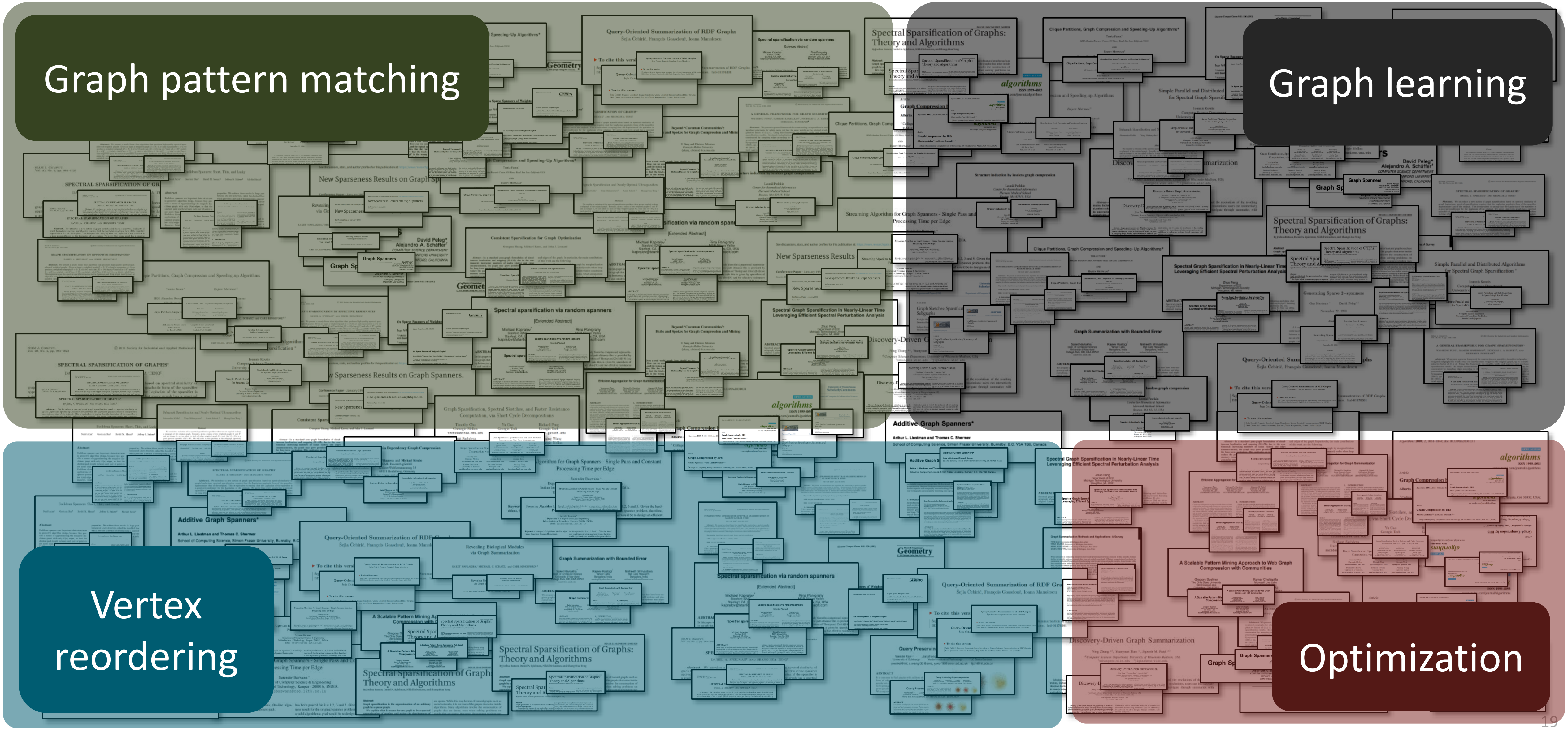
? **How** to **analyze** performance/others, using **what** metrics?

What are the representative problems & algorithms?

1

What are the representative problems & algorithms?

1



Graph pattern matching

Graph learning

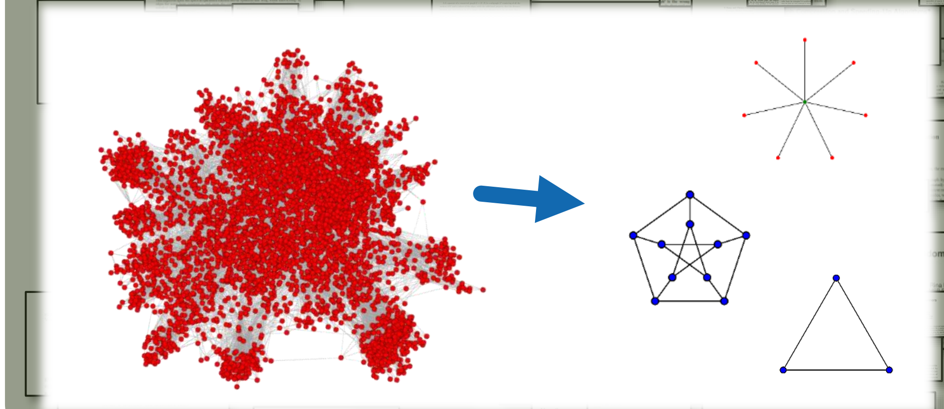
Vertex reordering

Optimization

What are the representative problems & algorithms?

1

Graph pattern matching



Graph learning

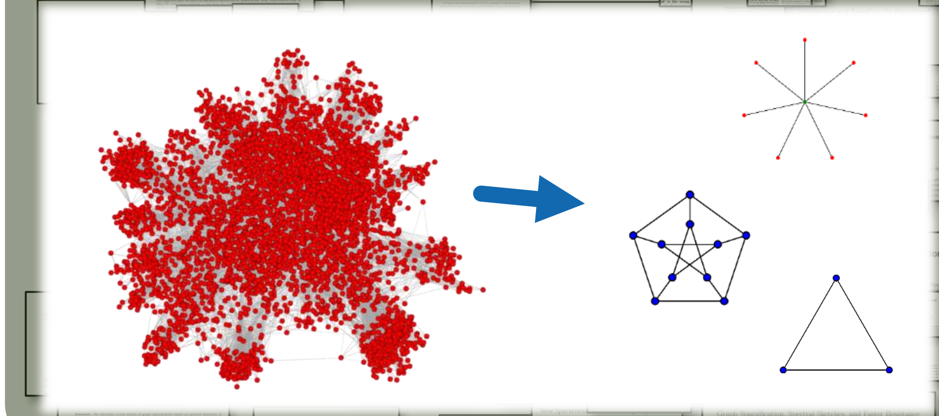
Vertex reordering

Optimization

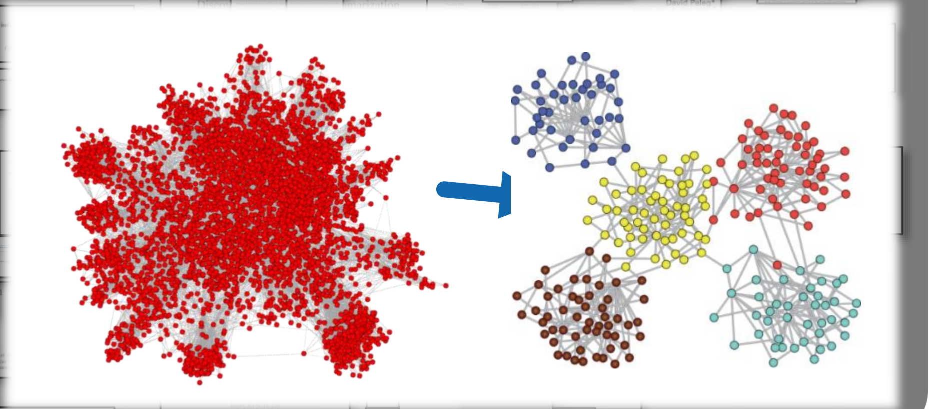
What are the representative problems & algorithms?

1

Graph pattern matching



Graph learning



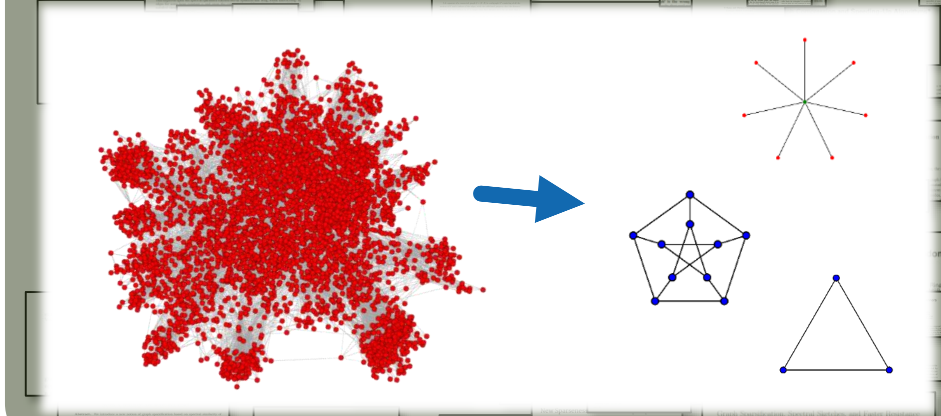
Vertex reordering

Optimization

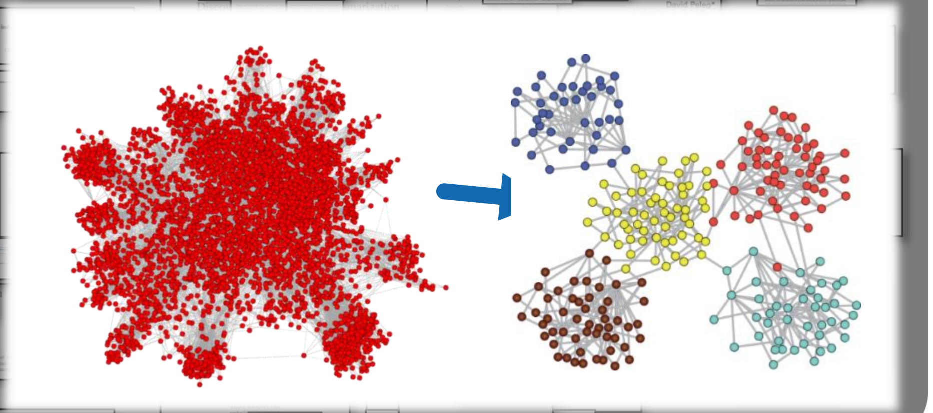
What are the representative problems & algorithms?

1

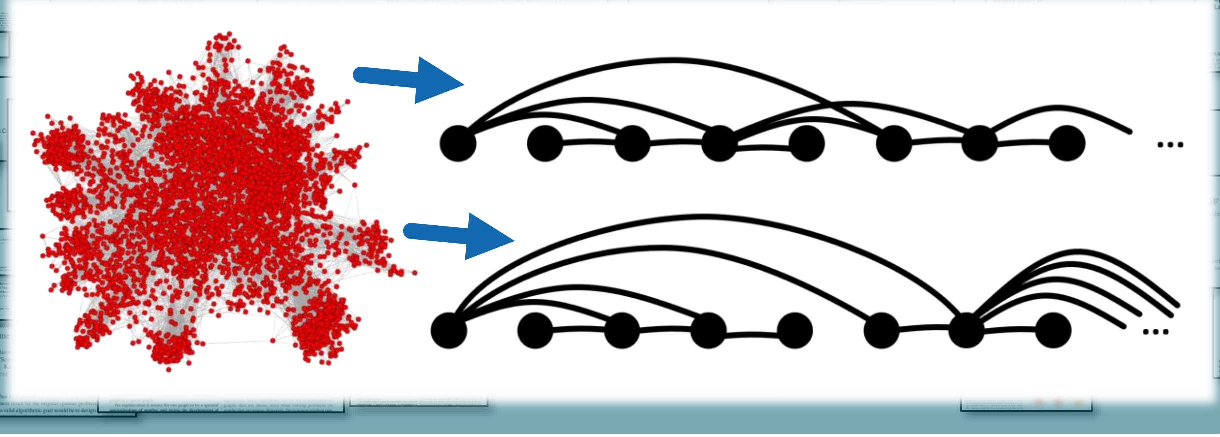
Graph pattern matching



Graph learning



Vertex reordering

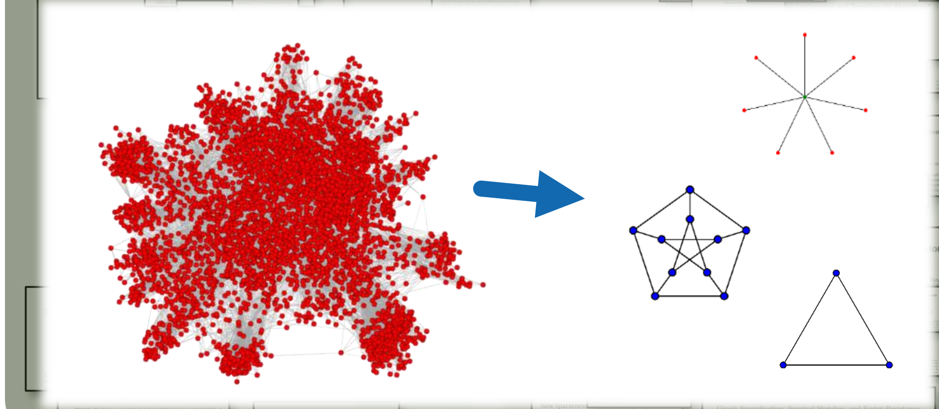


Optimization

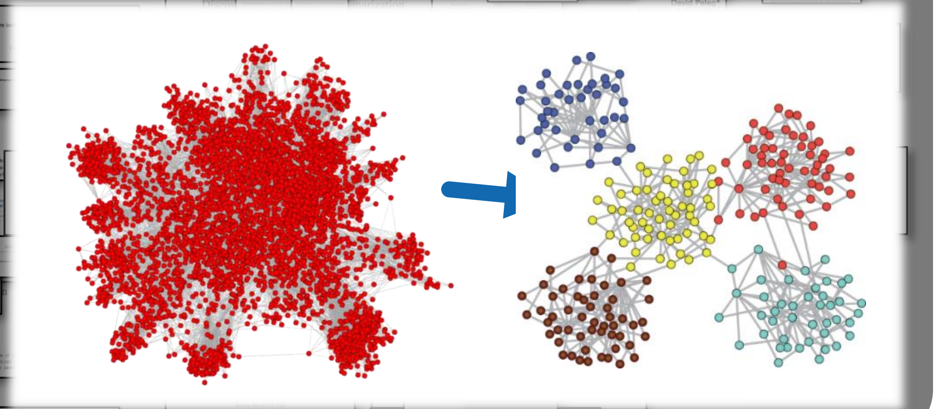
What are the representative problems & algorithms?

1

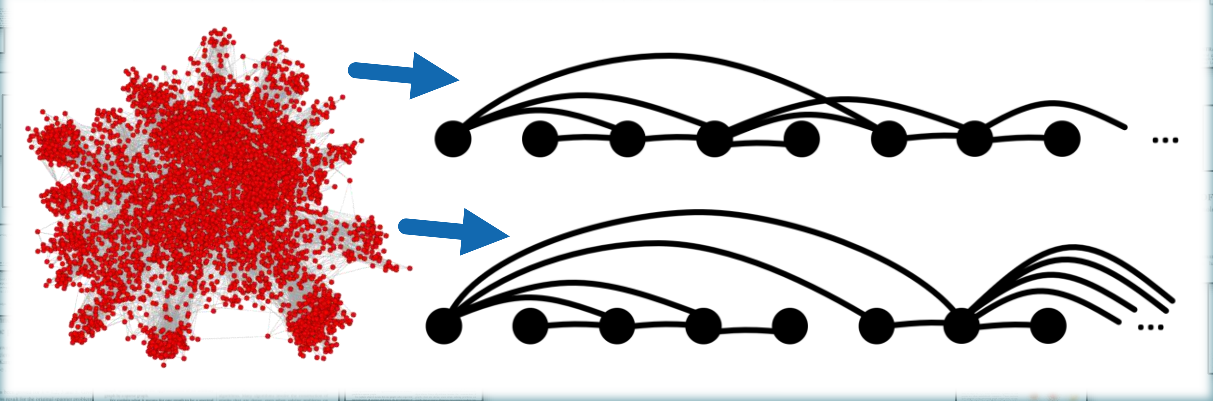
Graph pattern matching

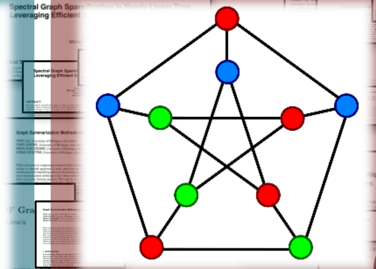


Graph learning



Vertex reordering





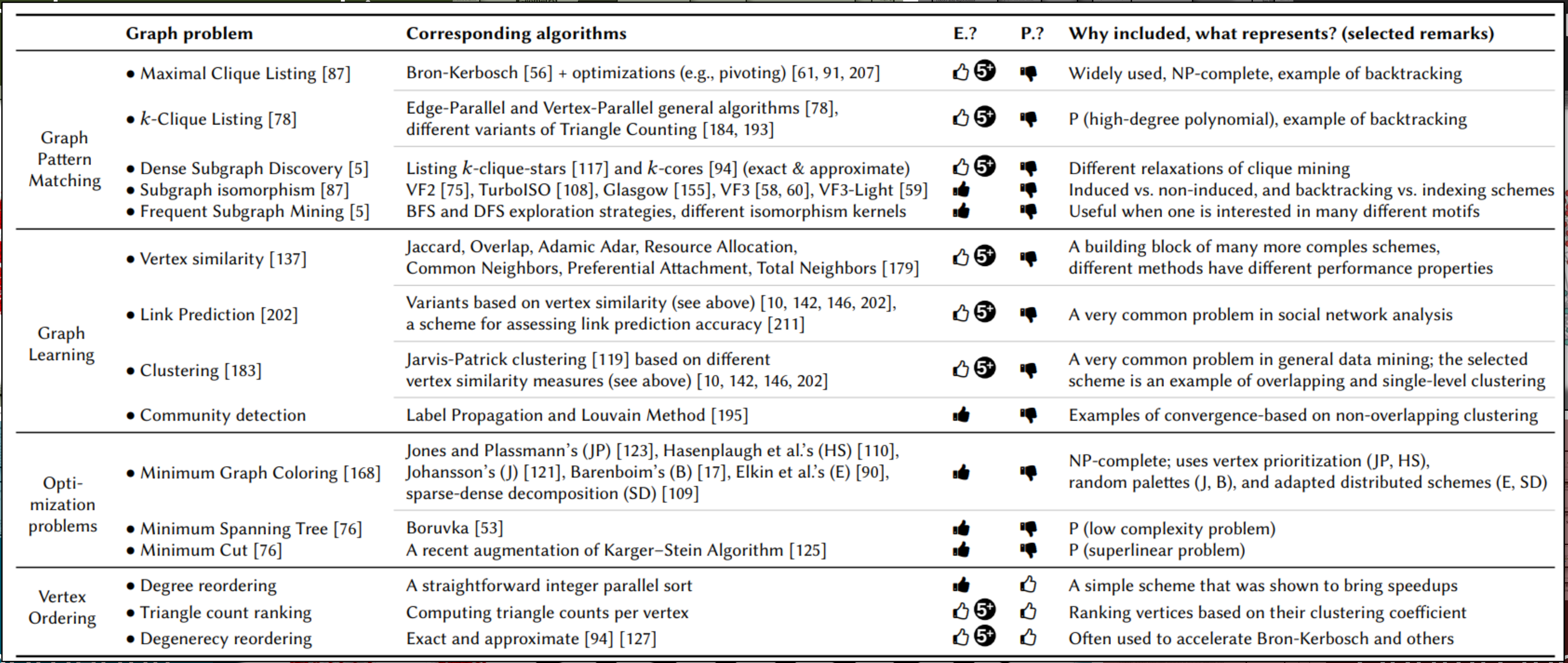
Optimization

What are the representative problems & algorithms? ...and datasets?

1

Gr

ng



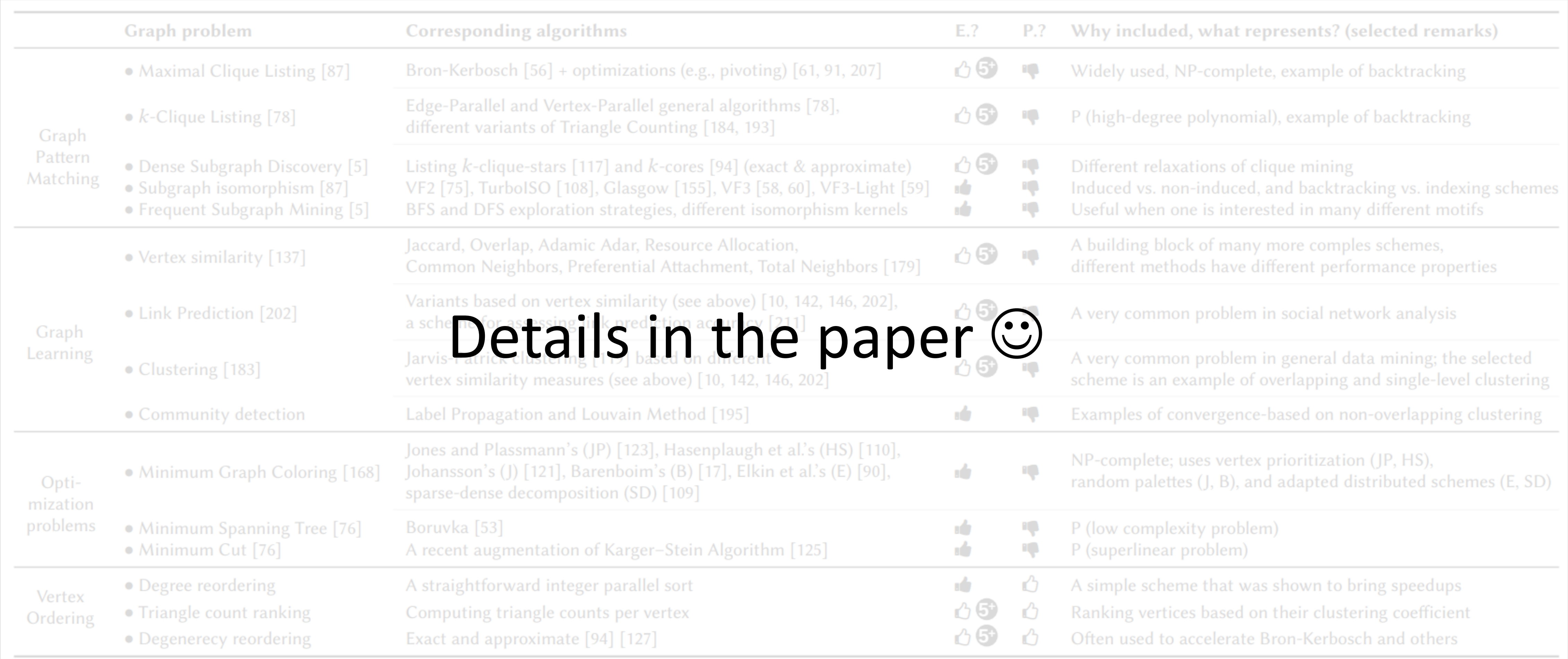
	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	🖒 5+	👎	Widely used, NP-complete, example of backtracking
	• k-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	🖒 5+	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing k-clique-stars [117] and k-cores [94] (exact & approximate)	🖒 5+	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Vertex similarity [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	🖒 5+	👎	A building block of many more comples schemes, different methods have different performance properties
	• Link Prediction [202]	Variants based on vertex similarity (see above) [10, 142, 146, 202], a scheme for assessing link prediction accuracy [211]	🖒 5+	👎	A very common problem in social network analysis
	• Clustering [183]	Jarvis-Patrick clustering [119] based on different vertex similarity measures (see above) [10, 142, 146, 202]	🖒 5+	👎	A very common problem in general data mining; the selected scheme is an example of overlapping and single-level clustering
	• Community detection	Label Propagation and Louvain Method [195]	👍	👎	Examples of convergence-based on non-overlapping clustering
Opti-mization problems	• Minimum Graph Coloring [168]	Jones and Plassmann's (JP) [123], Hasenplaugh et al.'s (HS) [110], Johansson's (J) [121], Barenboim's (B) [17], Elkin et al.'s (E) [90], sparse-dense decomposition (SD) [109]	👍	👎	NP-complete; uses vertex prioritization (JP, HS), random palettes (J, B), and adapted distributed schemes (E, SD)
	• Minimum Spanning Tree [76]	Boruvka [53]	👍	👎	P (low complexity problem)
	• Minimum Cut [76]	A recent augmentation of Karger–Stein Algorithm [125]	👍	👎	P (superlinear problem)
Vertex Ordering	• Degree reordering	A straightforward integer parallel sort	👍	🖒	A simple scheme that was shown to bring speedups
	• Triangle count ranking	Computing triangle counts per vertex	🖒 5+	🖒	Ranking vertices based on their clustering coefficient
	• Degenerecy reordering	Exact and approximate [94] [127]	🖒 5+	🖒	Often used to accelerate Bron-Kerbosch and others

reordering

Optimization

What are the representative problems & algorithms? ...and datasets?

1

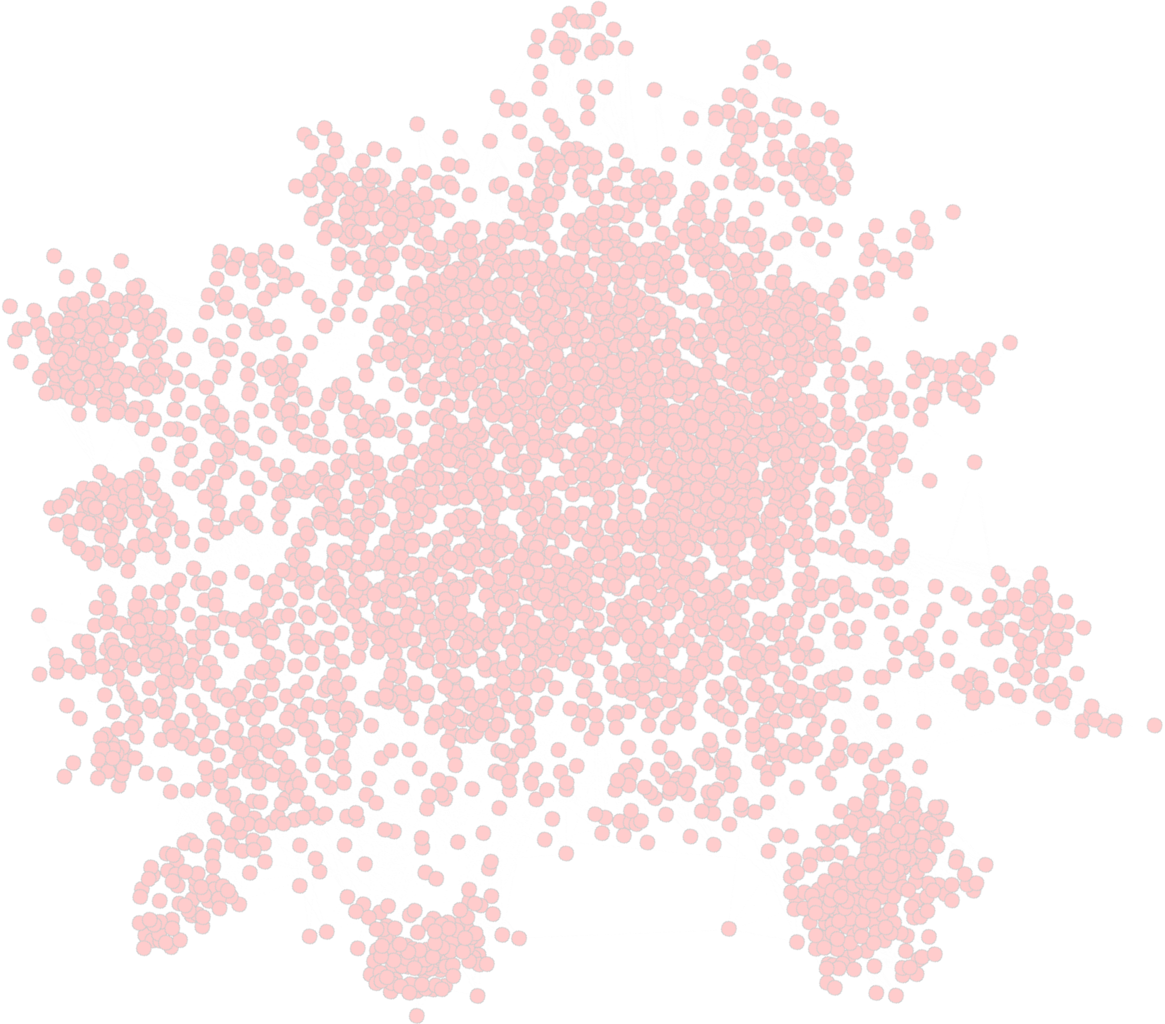


	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	🖒 5+	👎	Widely used, NP-complete, example of backtracking
	• *k*-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	🖒 5+	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing *k*-clique-stars [117] and *k*-cores [94] (exact & approximate)	🖒 5+	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Vertex similarity [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	🖒 5+	👎	A building block of many more comples schemes, different methods have different performance properties
	• Link Prediction [202]	Variants based on vertex similarity (see above) [10, 142, 146, 202], a scheme for assessing link prediction accuracy [211]	🖒 5+	👎	A very common problem in social network analysis
	• Clustering [183]	Jarvis-Patrick clustering [119] based on different vertex similarity measures (see above) [10, 142, 146, 202]	🖒 5+	👎	A very common problem in general data mining; the selected scheme is an example of overlapping and single-level clustering
	• Community detection	Label Propagation and Louvain Method [195]	👍	👎	Examples of convergence-based on non-overlapping clustering
Opti-mization problems	• Minimum Graph Coloring [168]	Jones and Plassmann's (JP) [123], Hasenplaugh et al.'s (HS) [110], Johansson's (J) [121], Barenboim's (B) [17], Elkin et al.'s (E) [90], sparse-dense decomposition (SD) [109]	👍	👎	NP-complete; uses vertex prioritization (JP, HS), random palettes (J, B), and adapted distributed schemes (E, SD)
	• Minimum Spanning Tree [76]	Boruvka [53]	👍	👎	P (low complexity problem)
	• Minimum Cut [76]	A recent augmentation of Karger–Stein Algorithm [125]	👍	👎	P (superlinear problem)
Vertex Ordering	• Degree reordering	A straightforward integer parallel sort	👍	🖒	A simple scheme that was shown to bring speedups
	• Triangle count ranking	Computing triangle counts per vertex	🖒 5+	🖒	Ranking vertices based on their clustering coefficient
	• Degenerecy reordering	Exact and approximate [94] [127]	🖒 5+	🖒	Often used to accelerate Bron-Kerbosch and others

Details in the paper ☺

How about datasets?

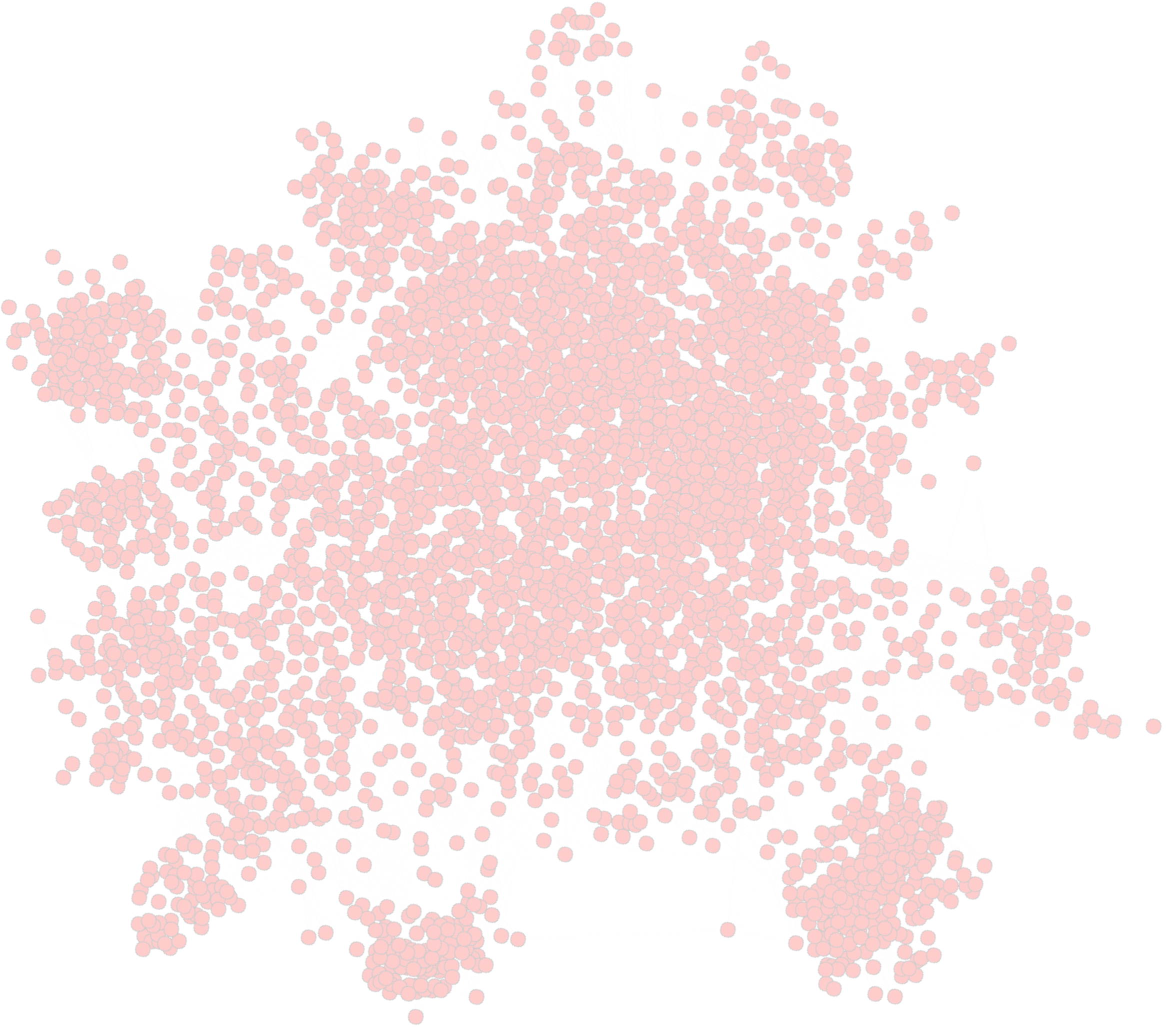
1



How about datasets?

When benchmarking graph workloads, one picks graphs with different…

1

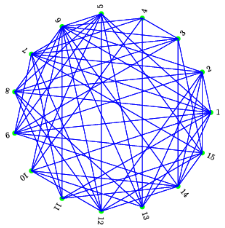


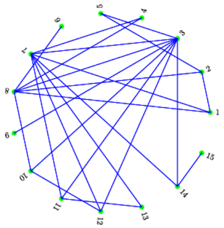
How about datasets?

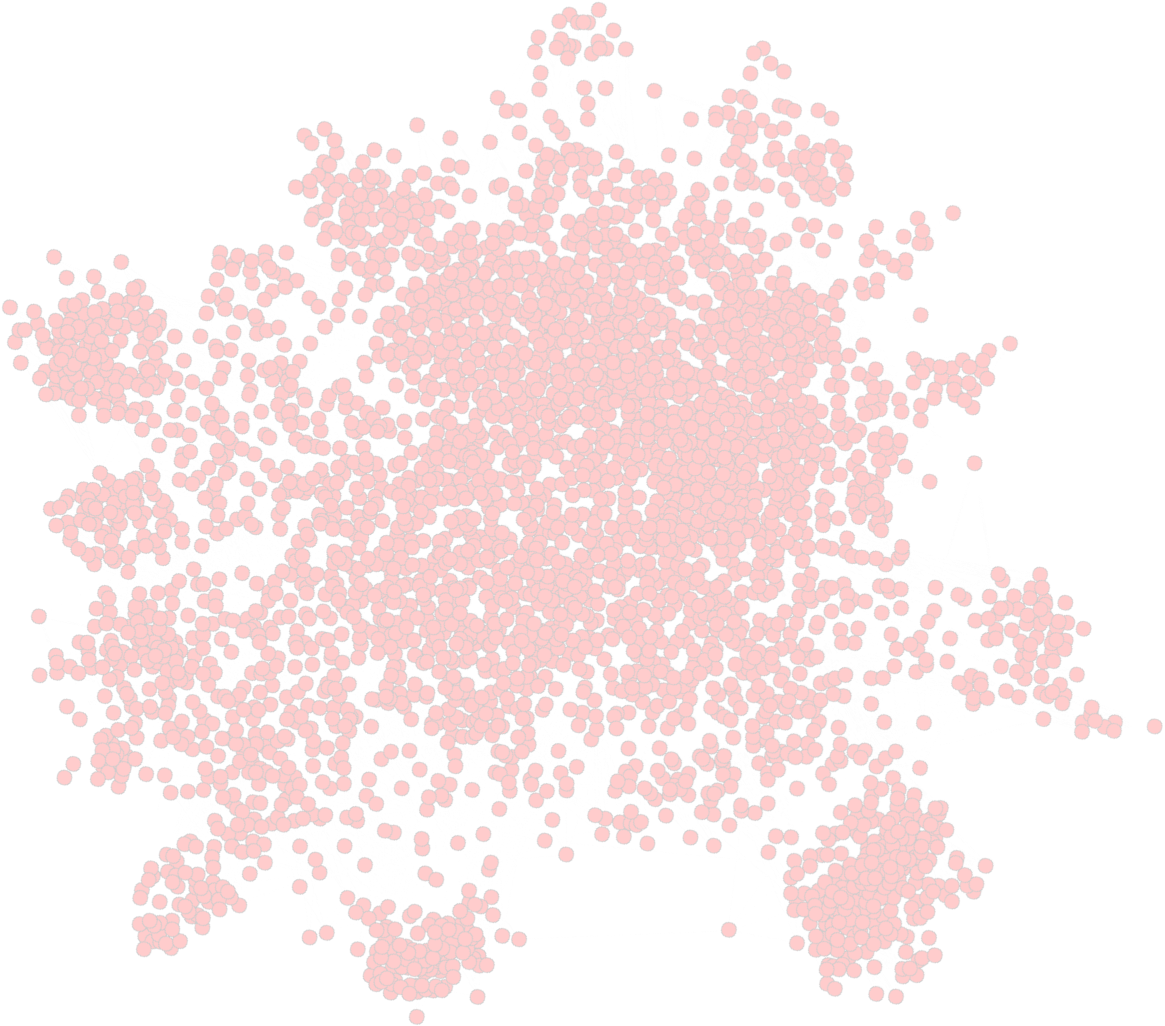
When benchmarking graph workloads, one picks graphs with different…

1

…Sparsities (a)







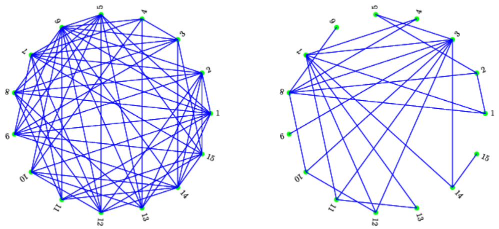
How about datasets?

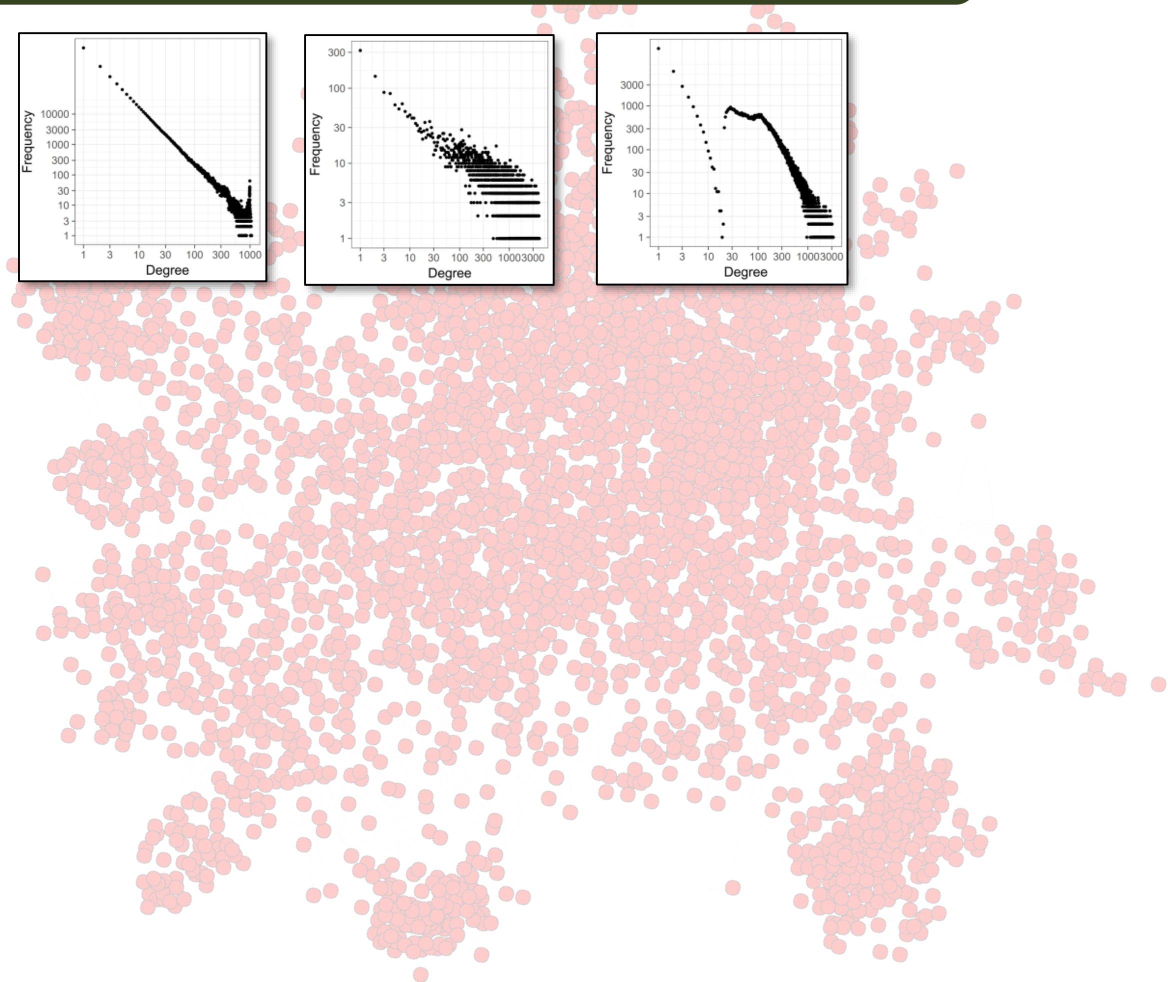
When benchmarking graph workloads, one picks graphs with different…

1

…Sparsities (a)

…Skews in degree distribution (b)





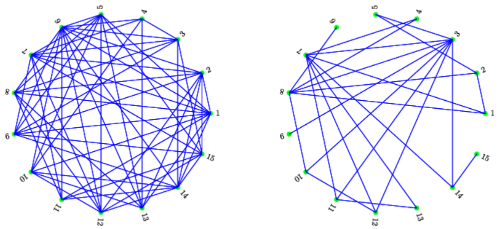
How about datasets?

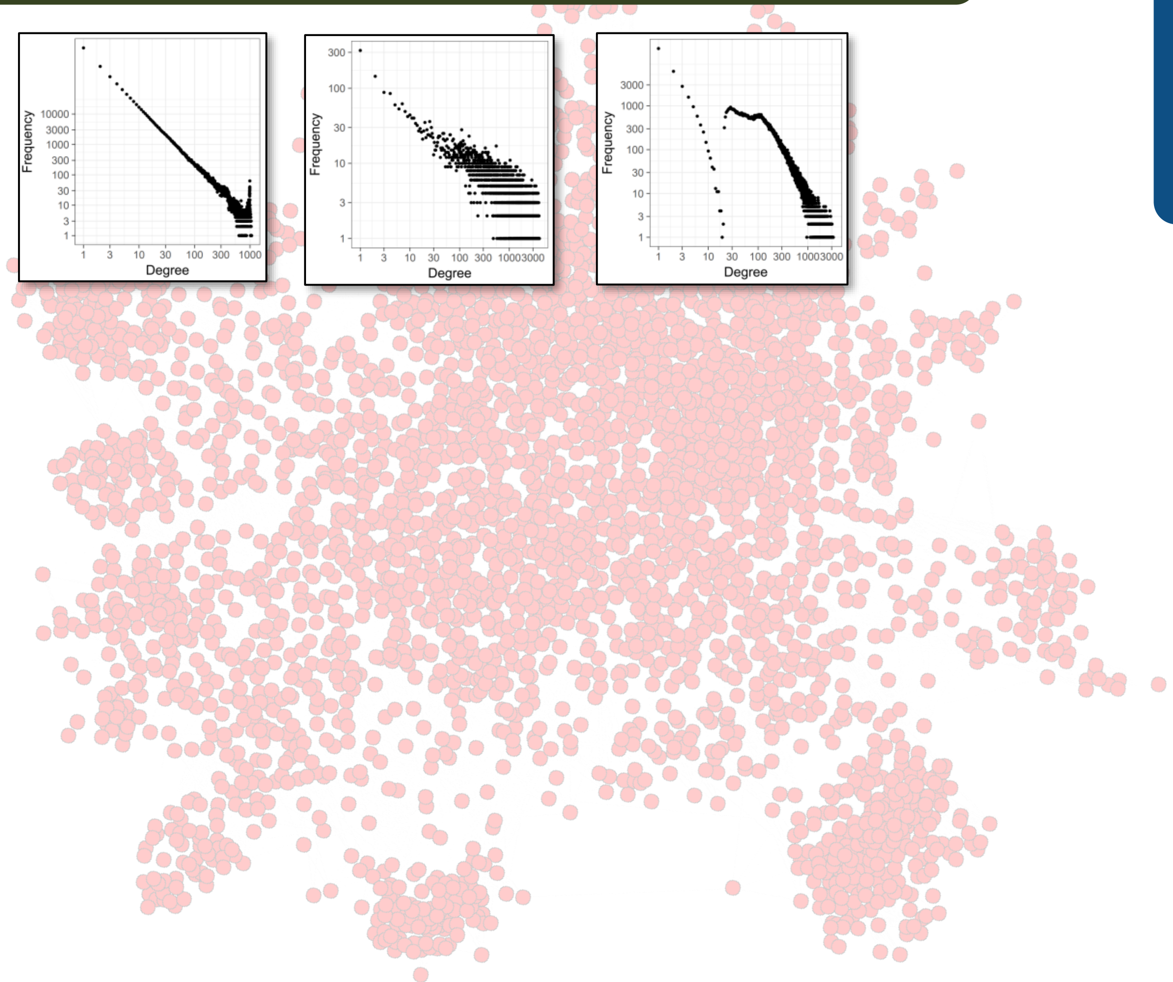
When benchmarking graph workloads, one picks graphs with different…

1

…Sparsities (a)

…Skews in degree distribution (b)





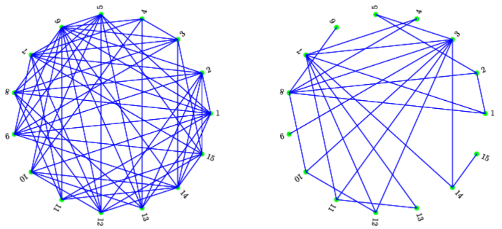
Not enough for graph mining!

How about datasets?

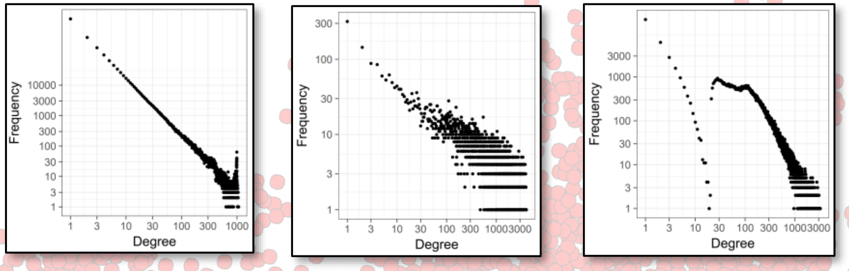
When benchmarking graph workloads, one picks graphs with different…

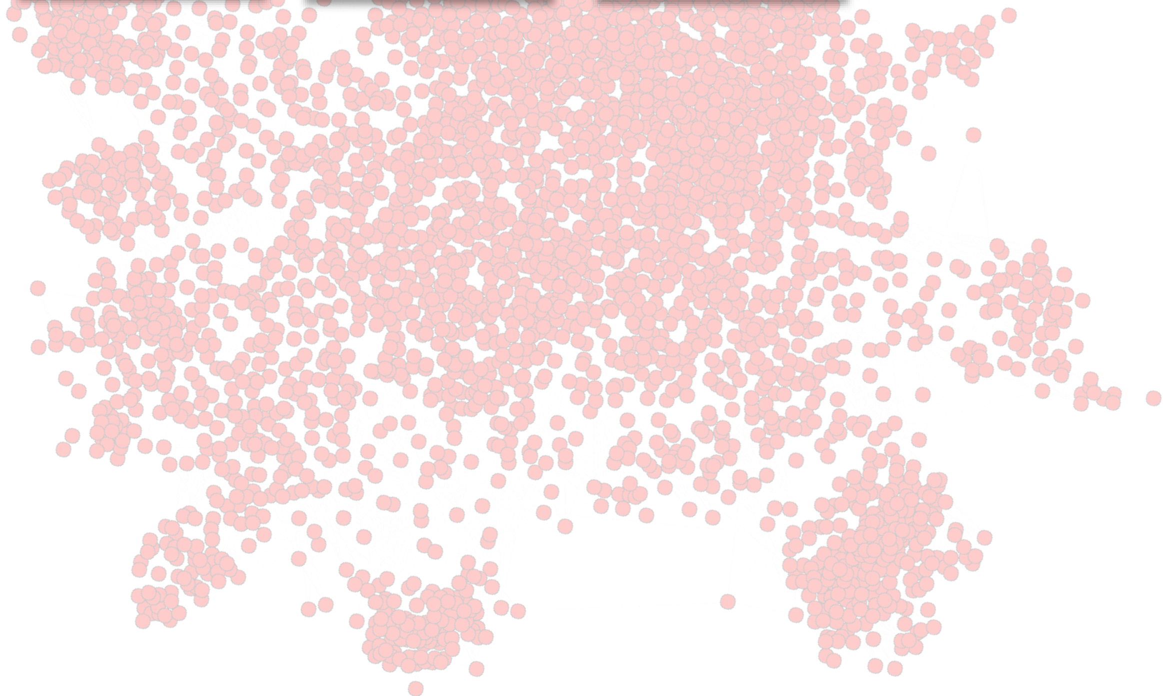
1

…Sparsities (a)



…Skews in degree distribution (b)





Not enough for graph mining!

Higher order organization matters

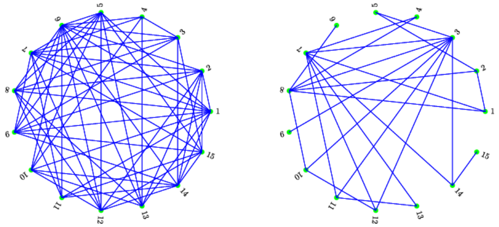
!

How about datasets?

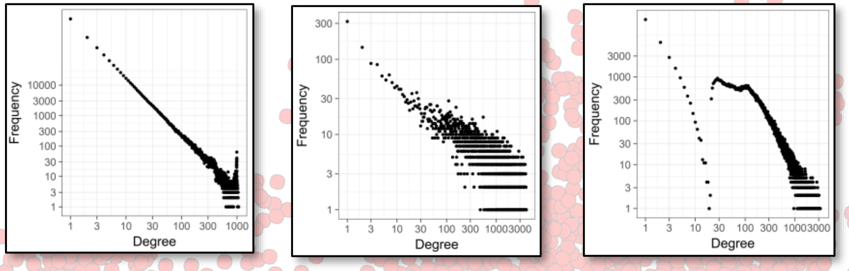
When benchmarking graph workloads, one picks graphs with different…

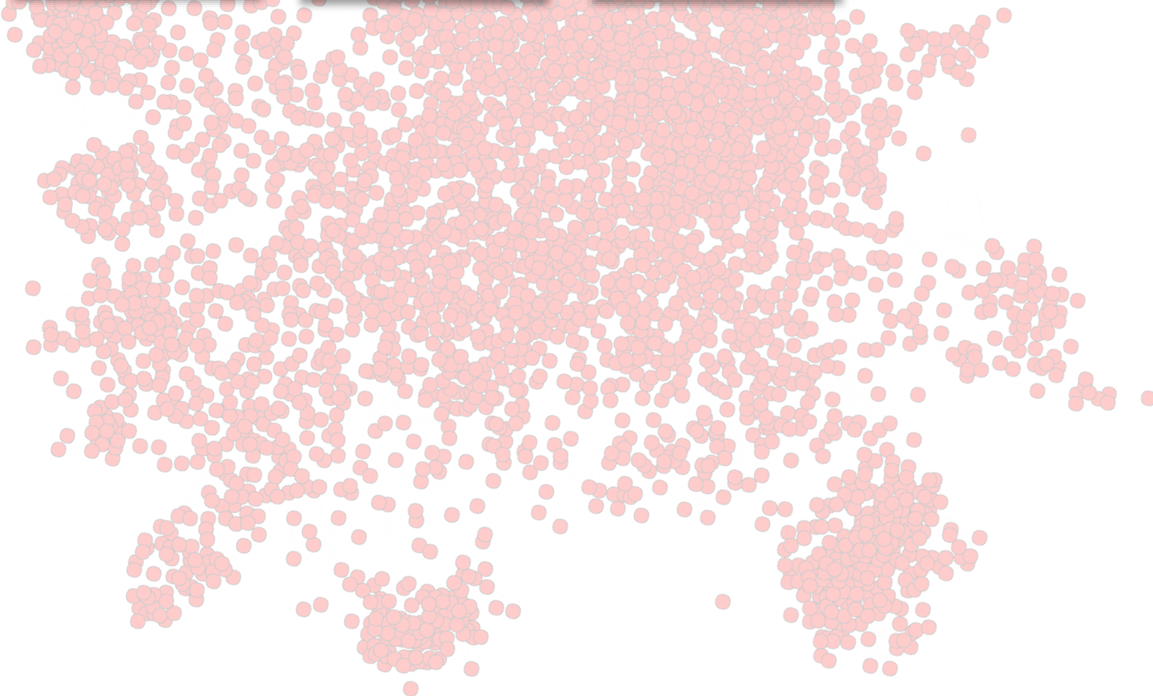
1

…Sparsities (a)



…Skews in degree distribution (b)

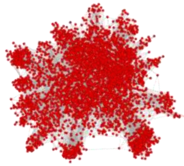




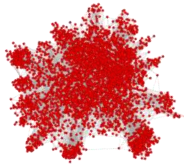
Not enough for graph mining!

Higher order organization matters

!



Flickr (F) photo relation graph



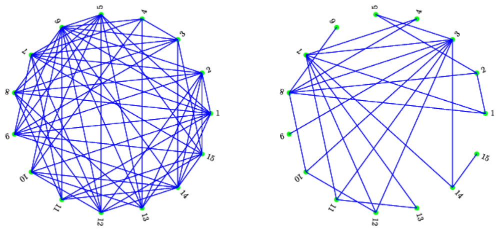
Livemocha (L) social network

How about datasets?

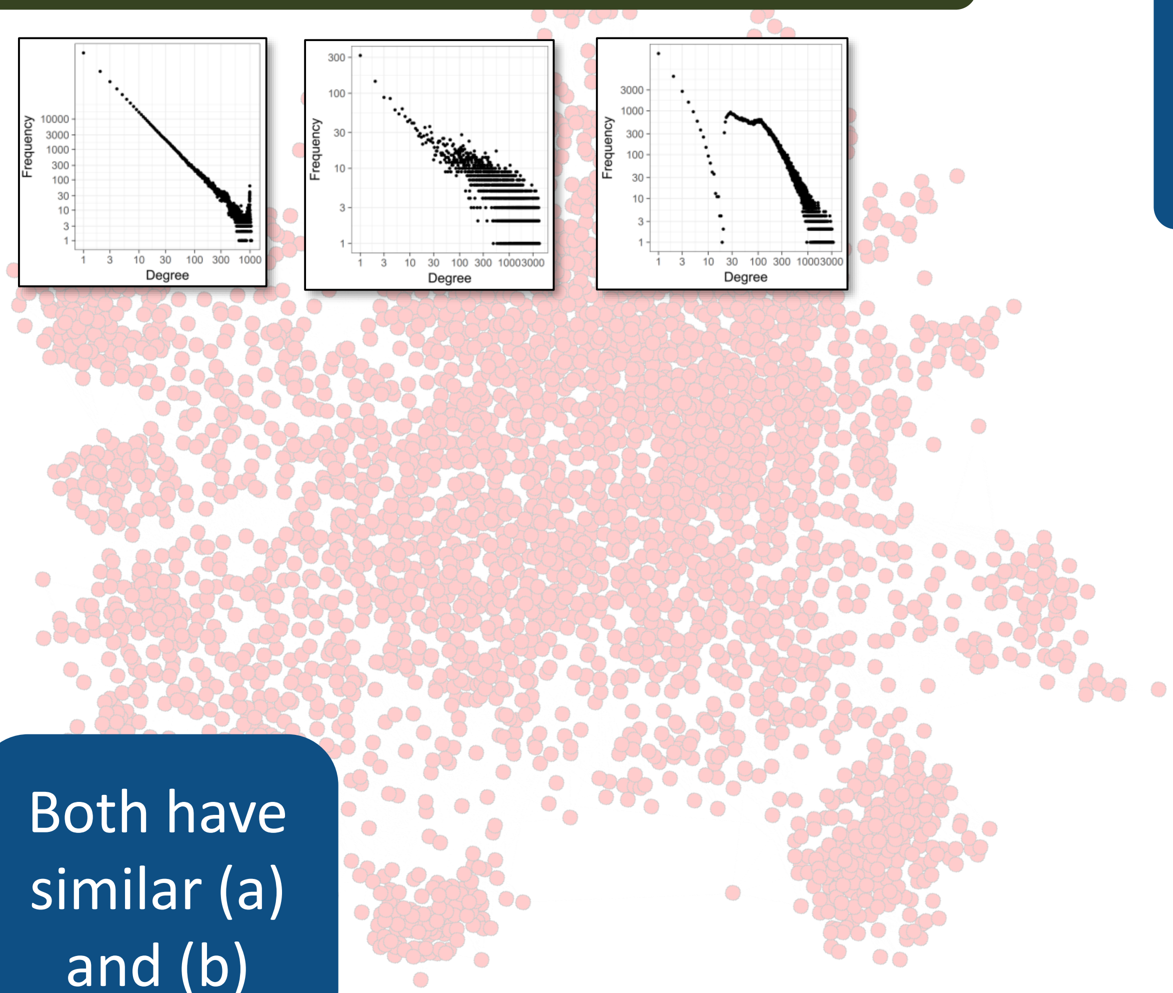
When benchmarking graph workloads, one picks graphs with different…

1

…Sparsities (a)



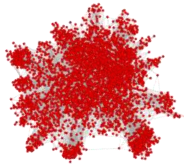
…Skews in degree distribution (b)



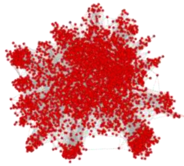
Not enough for graph mining!

Higher order organization matters

!



Flickr (F) photo relation graph



Livemocha (L) social network

→

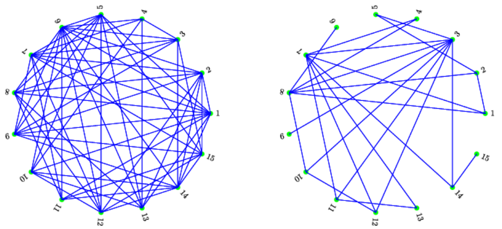
Both have similar (a) and (b)

How about datasets?

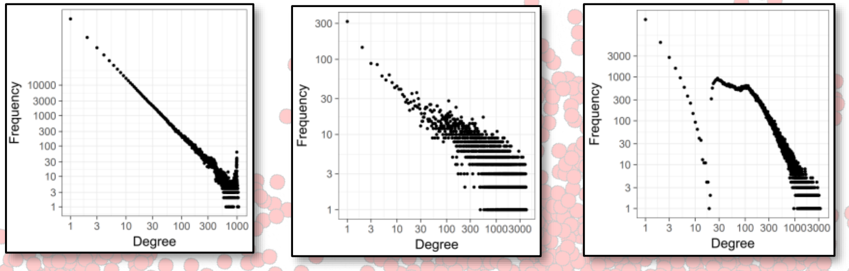
When benchmarking graph workloads, one picks graphs with different…

1

…Sparsities (a)

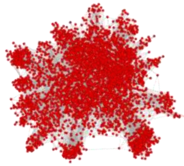


…Skews in degree distribution (b)

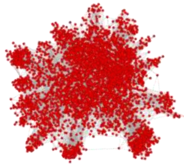


Not enough for graph mining!

Higher order organization matters !



Flickr (F) photo relation graph



Livemocha (L) social network

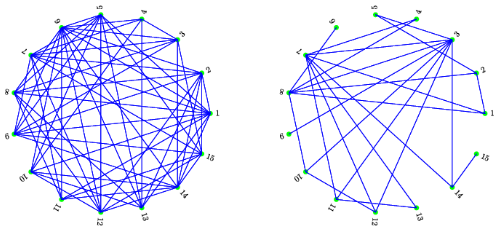
→ Both have similar (a) and (b) → Yet, (L) has 4.4M 4-cliques, and (F) has 9.6B 4-cliques

How about datasets?

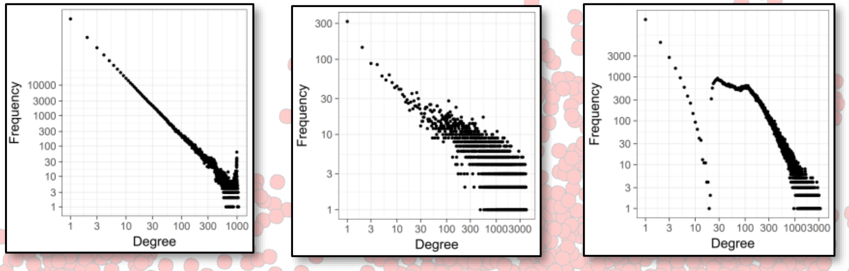
When benchmarking graph workloads, one picks graphs with different…

1

…Sparsities (a)



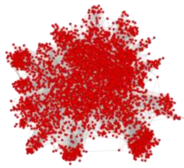
…Skews in degree distribution (b)



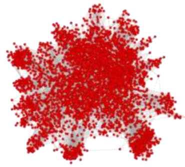
Not enough for graph mining!

Higher order organization matters

!



Flickr (F) photo relation graph



Livemocha (L) social network

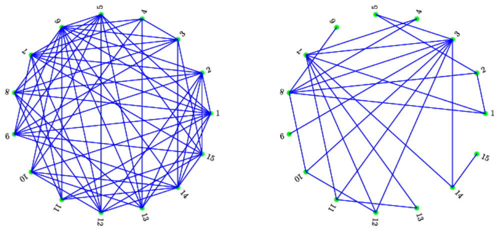
→ Both have similar (a) and (b) → Yet, (L) has 4.4M 4-cliques, and (F) has 9.6B 4-cliques → Different performance characteristics for mining problems

How about datasets?

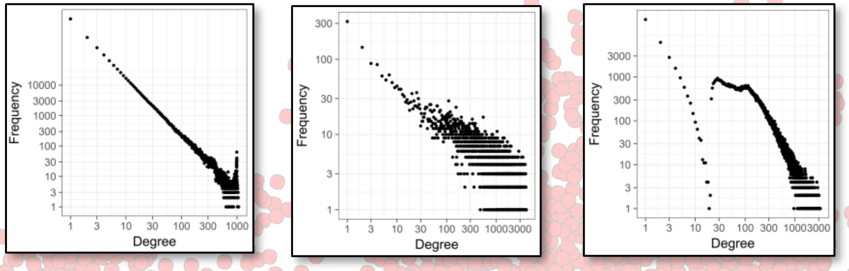
When benchmarking graph workloads, one picks graphs with different…

1

…Sparsities (a)



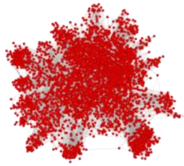
…Skews in degree distribution (b)



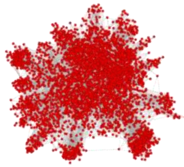
Not enough for graph mining!

Higher order organization matters !

…Differences in „complex structure” (e.g., #triangles per vertex)



Flickr (F) photo relation graph



Livemocha (L) social network

→ Both have similar (a) and (b)

→ Yet, (L) has <u>4.4M</u> 4-cliques, and (F) has <u>9.6B</u> 4-cliques

→ Different performance characteristics for mining problems

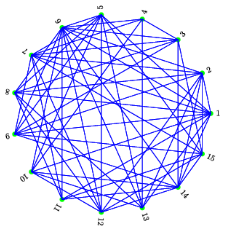
How about datasets?

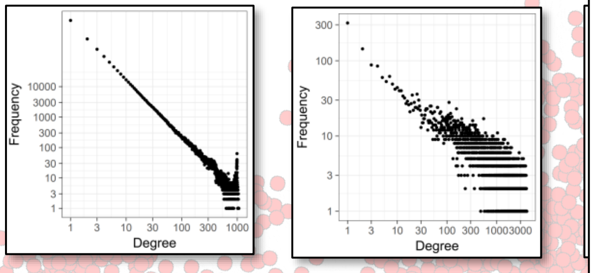
When benchmarking graph workloads, one picks graphs with different…

1

…Sparsities (a)

…Skews in degree distribution (b)

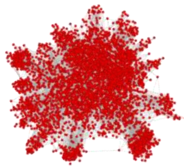




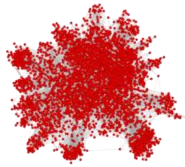
Higher order organization matters !

…Differences

#t



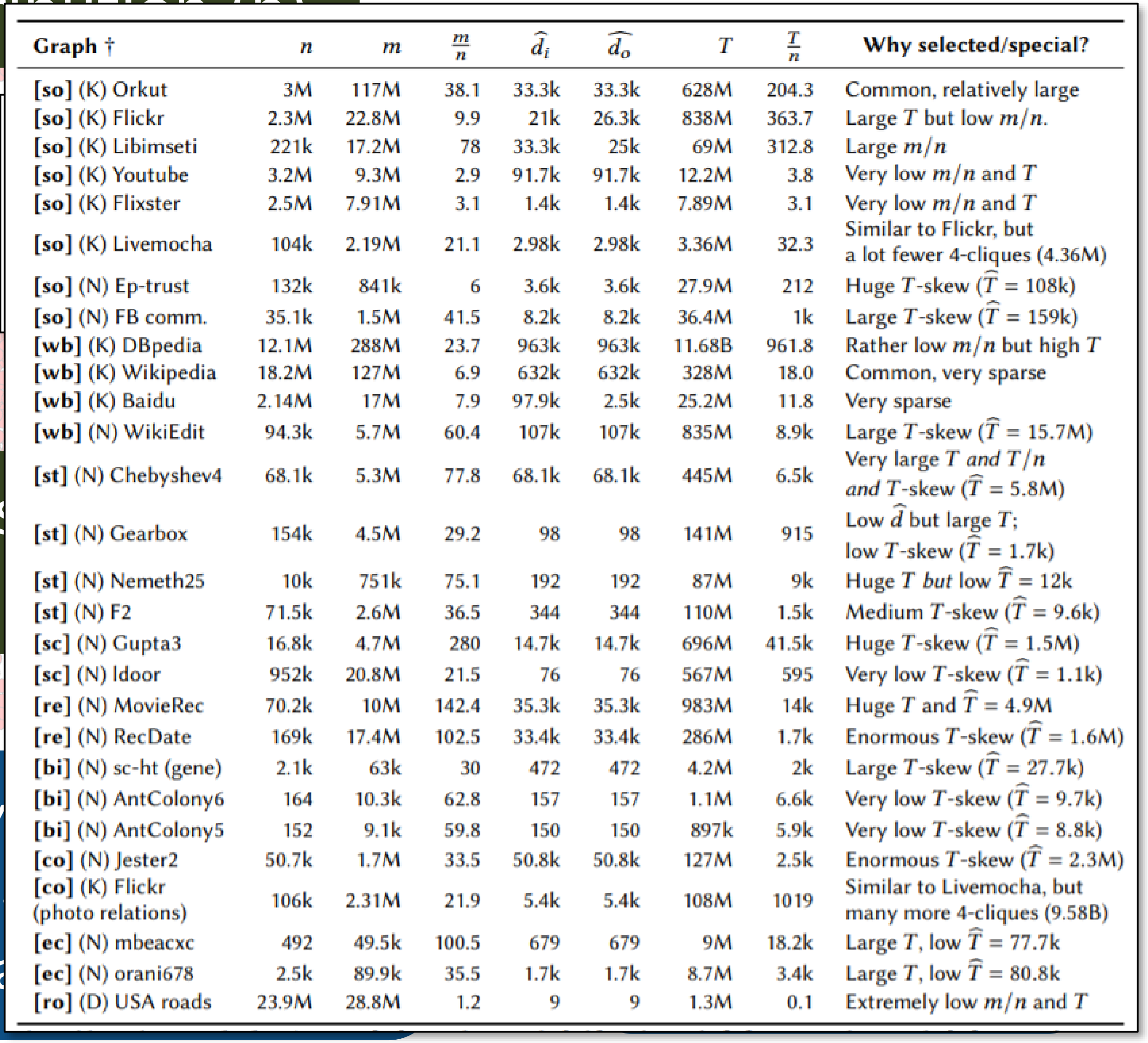
Flickr (F) photo relation graph



Livemocha (L) social network

Both have similar (a) and (b)

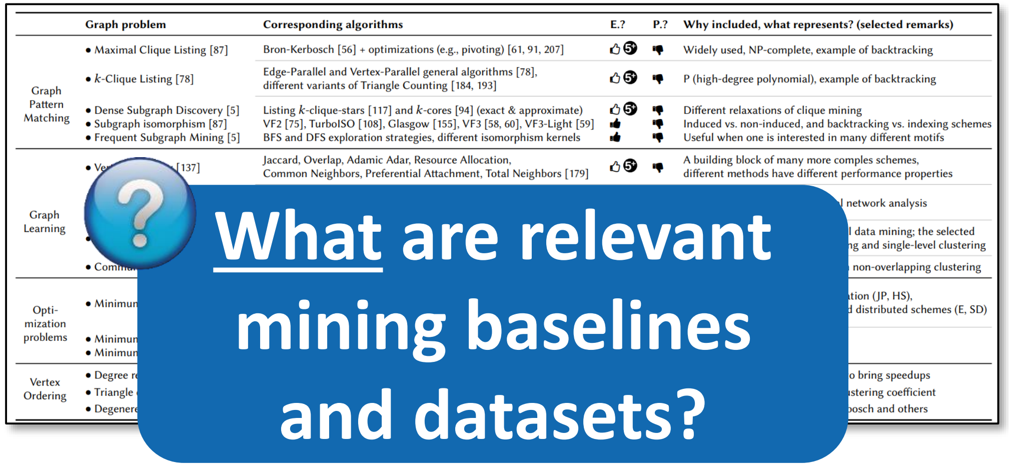
Y
4
ha



Graph †	n	m	$\frac{m}{n}$	$\widehat{d_i}$	$\widehat{d_o}$	T	$\frac{T}{n}$	Why selected/special?
[so] (K) Orkut	3M	117M	38.1	33.3k	33.3k	628M	204.3	Common, relatively large
[so] (K) Flickr	2.3M	22.8M	9.9	21k	26.3k	838M	363.7	Large T but low m/n.
[so] (K) Libimseti	221k	17.2M	78	33.3k	25k	69M	312.8	Large m/n
[so] (K) Youtube	3.2M	9.3M	2.9	91.7k	91.7k	12.2M	3.8	Very low m/n and T
[so] (K) Flixster	2.5M	7.91M	3.1	1.4k	1.4k	7.89M	3.1	Very low m/n and T
[so] (K) Livemocha	104k	2.19M	21.1	2.98k	2.98k	3.36M	32.3	Similar to Flickr, but a lot fewer 4-cliques (4.36M)
[so] (N) Ep-trust	132k	841k	6	3.6k	3.6k	27.9M	212	Huge T-skew ($\widehat{T}$ = 108k)
[so] (N) FB comm.	35.1k	1.5M	41.5	8.2k	8.2k	36.4M	1k	Large T-skew ($\widehat{T}$ = 159k)
[wb] (K) DBpedia	12.1M	288M	23.7	963k	963k	11.68B	961.8	Rather low m/n but high T
[wb] (K) Wikipedia	18.2M	127M	6.9	632k	632k	328M	18.0	Common, very sparse
[wb] (K) Baidu	2.14M	17M	7.9	97.9k	2.5k	25.2M	11.8	Very sparse
[wb] (N) WikiEdit	94.3k	5.7M	60.4	107k	107k	835M	8.9k	Large T-skew ($\widehat{T}$ = 15.7M)
[st] (N) Chebyshev4	68.1k	5.3M	77.8	68.1k	68.1k	445M	6.5k	Very large T *and* T/n *and* T-skew ($\widehat{T}$ = 5.8M)
[st] (N) Gearbox	154k	4.5M	29.2	98	98	141M	915	Low $\widehat{d}$ but large T; low T-skew ($\widehat{T}$ = 1.7k)
[st] (N) Nemeth25	10k	751k	75.1	192	192	87M	9k	Huge T *but* low $\widehat{T}$ = 12k
[st] (N) F2	71.5k	2.6M	36.5	344	344	110M	1.5k	Medium T-skew ($\widehat{T}$ = 9.6k)
[sc] (N) Gupta3	16.8k	4.7M	280	14.7k	14.7k	696M	41.5k	Huge T-skew ($\widehat{T}$ = 1.5M)
[sc] (N) Idoor	952k	20.8M	21.5	76	76	567M	595	Very low T-skew ($\widehat{T}$ = 1.1k)
[re] (N) MovieRec	70.2k	10M	142.4	35.3k	35.3k	983M	14k	Huge T and $\widehat{T}$ = 4.9M
[re] (N) RecDate	169k	17.4M	102.5	33.4k	33.4k	286M	1.7k	Enormous T-skew ($\widehat{T}$ = 1.6M)
[bi] (N) sc-ht (gene)	2.1k	63k	30	472	472	4.2M	2k	Large T-skew ($\widehat{T}$ = 27.7k)
[bi] (N) AntColony6	164	10.3k	62.8	157	157	1.1M	6.6k	Very low T-skew ($\widehat{T}$ = 9.7k)
[bi] (N) AntColony5	152	9.1k	59.8	150	150	897k	5.9k	Very low T-skew ($\widehat{T}$ = 8.8k)
[co] (N) Jester2	50.7k	1.7M	33.5	50.8k	50.8k	127M	2.5k	Enormous T-skew ($\widehat{T}$ = 2.3M)
[co] (K) Flickr (photo relations)	106k	2.31M	21.9	5.4k	5.4k	108M	1019	Similar to Livemocha, but many more 4-cliques (9.58B)
[ec] (N) mbeacxc	492	49.5k	100.5	679	679	9M	18.2k	Large T, low $\widehat{T}$ = 77.7k
[ec] (N) orani678	2.5k	89.9k	35.5	1.7k	1.7k	8.7M	3.4k	Large T, low $\widehat{T}$ = 80.8k
[ro] (D) USA roads	23.9M	28.8M	1.2	9	9	1.3M	0.1	Extremely low m/n and T

GraphMineSuite (GMS) comes with...

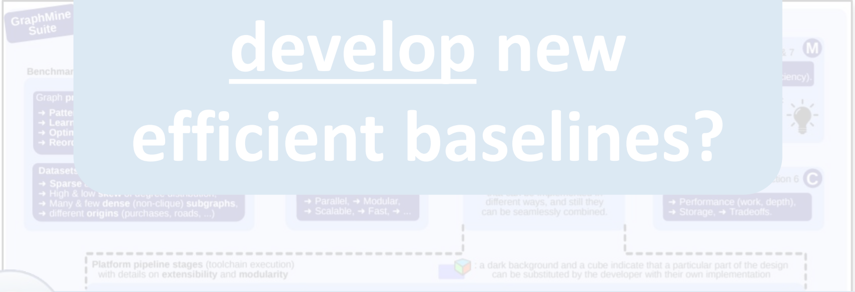
1 ... **Benchmark specification** prescribing representative *problems*, *algorithms*, and *datasets*



	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	👍 5+	👎	Widely used, NP-complete, example of backtracking
	• *k*-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	👍 5+	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing *k*-clique-stars [117] and *k*-cores [94] (exact & approximate)	👍 5+	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Ve [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	👍 5+	👎	A building block of many more comples schemes, different methods have different performance properties
					l network analysis
	•				l data mining; the selected ng and single-level clustering
	• Comm				non-overlapping clustering
Opti- mization problems	• Minimum				tion (JP, HS), d distributed schemes (E, SD)
	• Minimum • Minimum				
Vertex Ordering	• Degree r				o bring speedups
	• Triangle				stering coefficient
	• Degenere				bosch and others

What are relevant mining baselines and datasets?

2 ... Software **platform** with **reference implementations** based on **set algebraic formulations** for *programmability* and *high performance*


GraphMine Suite
Benchmark
Datasets
Platform pipeline stages (toolchain execution) with details on extensibility and modularity

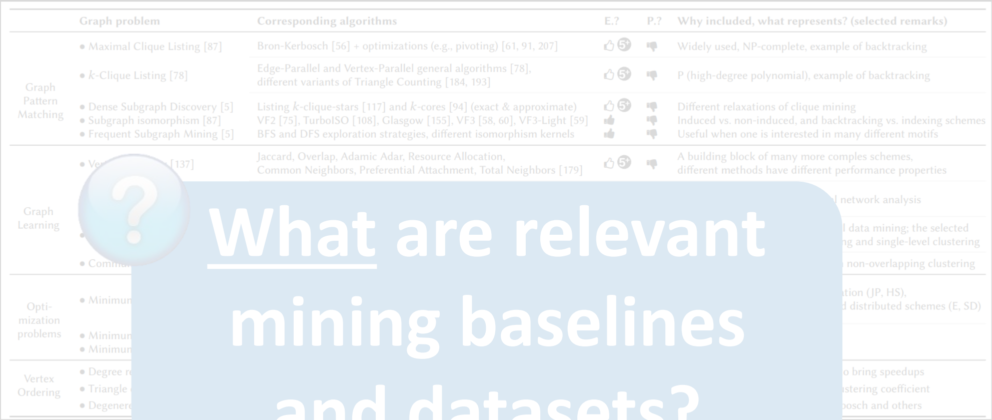
How to effectively **develop** new efficient baselines?

3 ... **Novel performance metric** that assesses *algorithmic throughput*

How to **analyze** performance/others, using **what** metrics?

GraphMineSuite (GMS) comes with...

1 ... **Benchmark specification** prescribing representative *problems*, *algorithms*, and *datasets*



	Graph problem	Corresponding algorithms	E.?	P.?	Why included, what represents? (selected remarks)
Graph Pattern Matching	• Maximal Clique Listing [87]	Bron-Kerbosch [56] + optimizations (e.g., pivoting) [61, 91, 207]	👍 5+	👎	Widely used, NP-complete, example of backtracking
	• *k*-Clique Listing [78]	Edge-Parallel and Vertex-Parallel general algorithms [78], different variants of Triangle Counting [184, 193]	👍 5+	👎	P (high-degree polynomial), example of backtracking
	• Dense Subgraph Discovery [5]	Listing *k*-clique-stars [117] and *k*-cores [94] (exact & approximate)	👍 5+	👎	Different relaxations of clique mining
	• Subgraph isomorphism [87]	VF2 [75], TurboISO [108], Glasgow [155], VF3 [58, 60], VF3-Light [59]	👍	👎	Induced vs. non-induced, and backtracking vs. indexing schemes
	• Frequent Subgraph Mining [5]	BFS and DFS exploration strategies, different isomorphism kernels	👍	👎	Useful when one is interested in many different motifs
Graph Learning	• Ve [137]	Jaccard, Overlap, Adamic Adar, Resource Allocation, Common Neighbors, Preferential Attachment, Total Neighbors [179]	👍 5+	👎	A building block of many more comples schemes, different methods have different performance properties
					l network analysis
					l data mining; the selected ng and single-level clustering
	• Comm				non-overlapping clustering
Opti-mization problems	• Minimum				tion (JP, HS), d distributed schemes (E, SD)
	• Minimum • Minimum				
Vertex Ordering	• Degree r				o bring speedups
	• Triangle				stering coefficient
	• Degenere				bosch and others

? **What** are relevant mining baselines and datasets?

2 ... Software **platform** with **reference implementations** based on **set algebraic formulations** for *programmability* and *high performance*

Graph pr
→ Patte
→ Learn
→ Optim
→ Reor

Datasets
→ Sparse
→ High & low
→ Many & few **dense** (non-clique) **subgraphs**,
→ different **origins** (purchases, roads, ...)

→ Parallel, → Modular,
→ Scalable, → Fast, → ...

different ways, and still they
can be seamlessly combined.

& 7

tion 6

→ Performance (work, depth),
→ Storage, → Tradeoffs.

Platform pipeline stages (toolchain execution)
with details on **extensibility** and **modularity**

: a dark background and a cube indicate that a particular part of the design can be substituted by the developer with their own implementation

? **How** to effectively **develop** new efficient baselines?

3 ... **Novel performance metric** that assesses *algorithmic throughput*

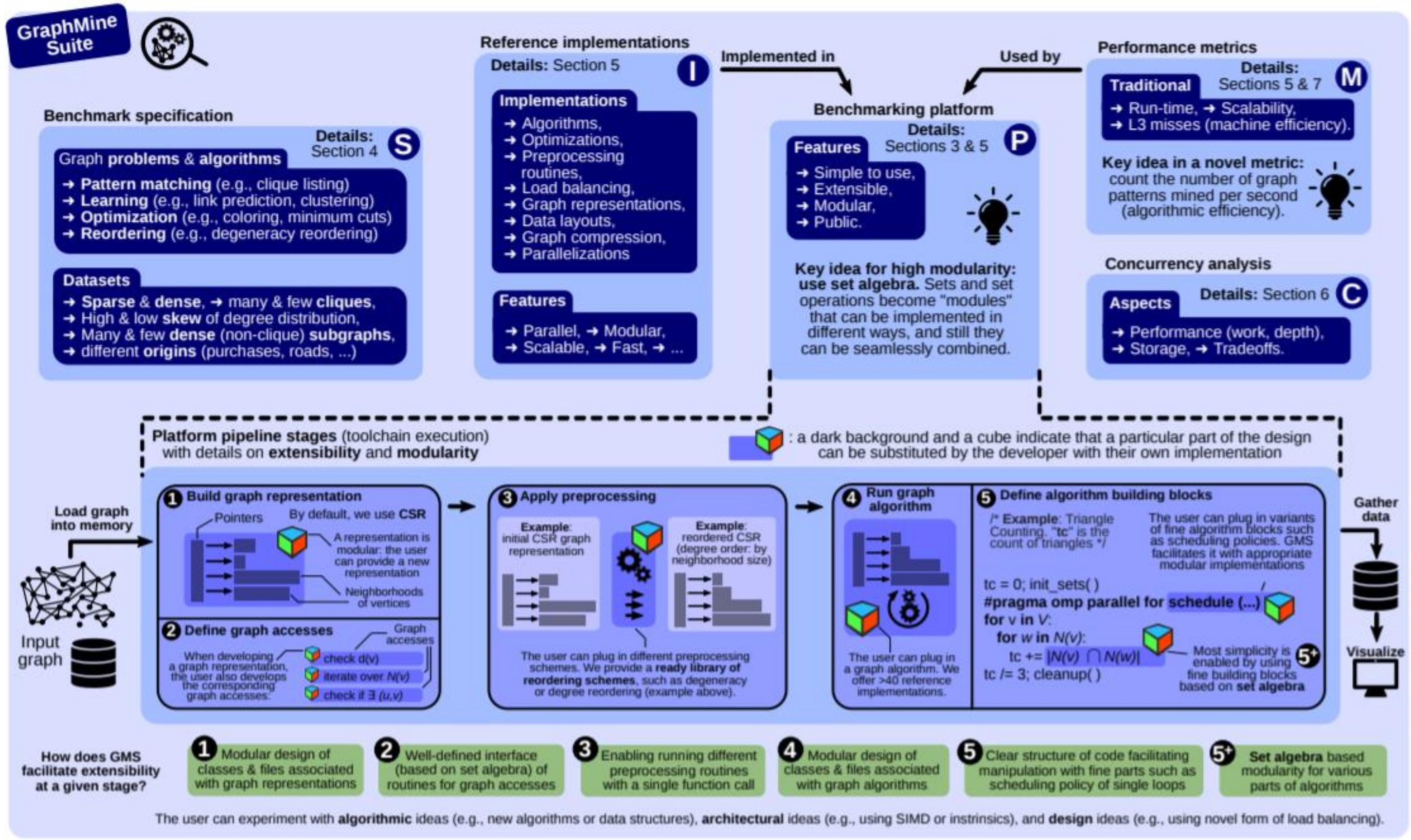
? **How** to **analyze** performance/others, using **what** metrics?

GMS software platform & reference implementations

2

GMS software platform & reference implementations

2



Benchmark specification

Details: Section 4

Graph **problems** & **algorithms**

- → **Pattern matching** (e.g., clique listing)
- → **Learning** (e.g., link prediction, clustering)
- → **Optimization** (e.g., coloring, minimum cuts)
- → **Reordering** (e.g., degeneracy reordering)

Datasets

- → **Sparse** & **dense**, → many & few **cliques**,
- → High & low **skew** of degree distribution,
- → Many & few **dense** (non-clique) **subgraphs**,
- → different **origins** (purchases, roads, ...)

Reference implementations

Details: Section 5

Implementations

- → Algorithms,
- → Optimizations,
- → Preprocessing routines,
- → Load balancing,
- → Graph representations,
- → Data layouts,
- → Graph compression,
- → Parallelizations

Features

- → Parallel, → Modular,
- → Scalable, → Fast, → ...

Benchmarking platform

Details: Sections 3 & 5

Features

- → Simple to use,
- → Extensible,
- → Modular,
- → Public.

Key idea for high modularity: use set algebra. Sets and set operations become "modules" that can be implemented in different ways, and still they can be seamlessly combined.

Performance metrics

Details: Sections 5 & 7

Traditional

- → Run-time, → Scalability,
- → L3 misses (machine efficiency).

Key idea in a novel metric: count the number of graph patterns mined per second (algorithmic efficiency).

Concurrency analysis

Details: Section 6

Aspects

- → Performance (work, depth),
- → Storage, → Tradeoffs.

Platform pipeline stages (toolchain execution) with details on **extensibility** and **modularity**

: a dark background and a cube indicate that a particular part of the design can be substituted by the developer with their own implementation

1 Build graph representation

By default, we use **CSR**

A representation is modular: the user can provide a new representation

Neighborhoods of vertices

2 Define graph accesses

When developing a graph representation, the user also develops the corresponding graph accesses:

- check d(v)
- iterate over *N(v)*
- check if ∃ *(u,v)*

3 Apply preprocessing

Example: initial CSR graph representation

Example: reordered CSR (degree order: by neighborhood size)

The user can plug in different preprocessing schemes. We provide a **ready library of reordering schemes**, such as degeneracy or degree reordering (example above).

4 Run graph algorithm

The user can plug in a graph algorithm. We offer >40 reference implementations.

5 Define algorithm building blocks

```
/* Example: Triangle
Counting. "tc" is the
count of triangles */

tc = 0; init_sets( )
#pragma omp parallel for schedule (...)
for v in V:
  for w in N(v):
    tc += |N(v) ∩ N(w)|
tc /= 3; cleanup( )
```

The user can plug in variants of fine algorithm blocks such as scheduling policies. GMS facilitates it with appropriate modular implementations

Most simplicity is enabled by using fine building blocks based on **set algebra** 5+

How does GMS facilitate extensibility at a given stage?

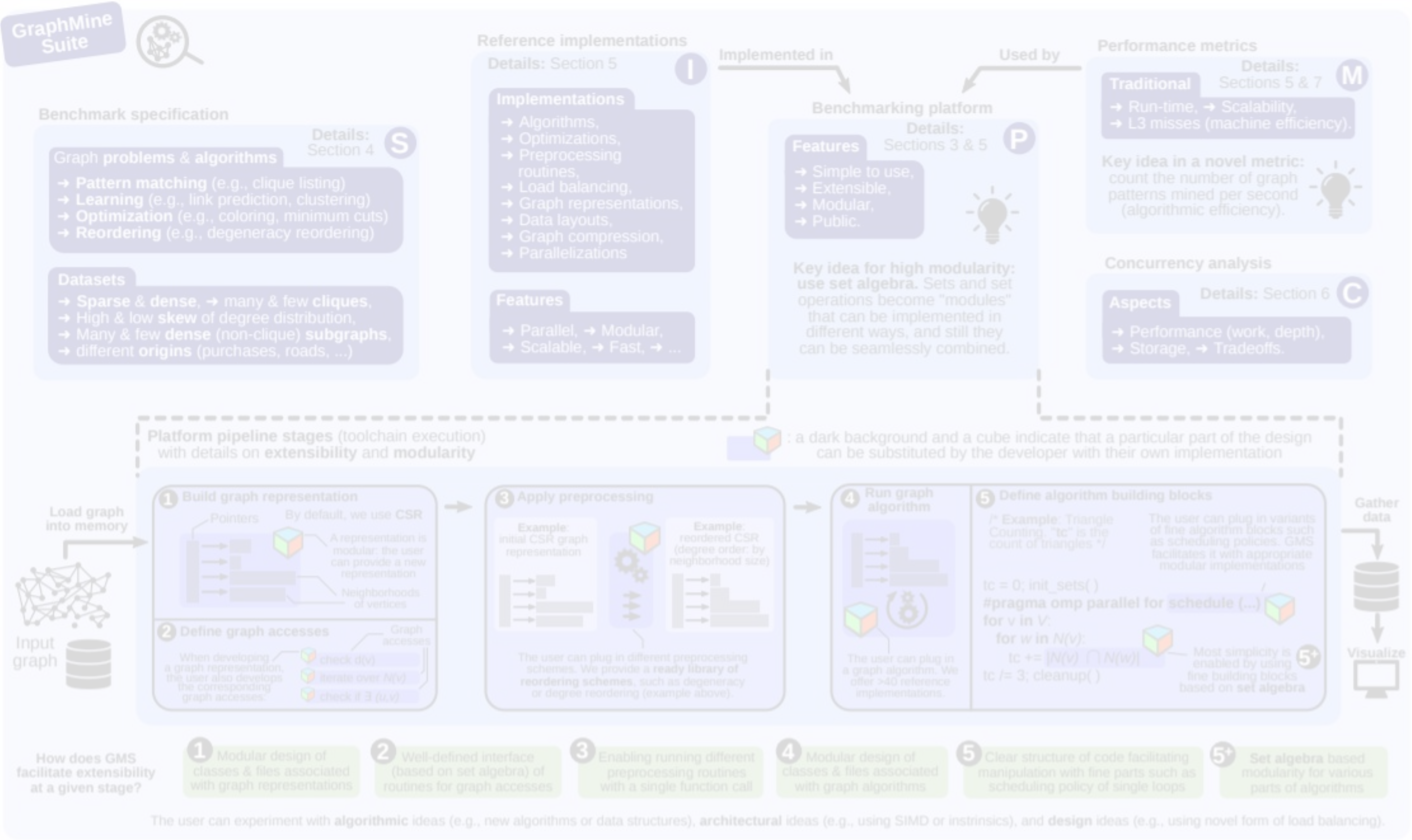
1. Modular design of classes & files associated with graph representations
2. Well-defined interface (based on set algebra) of routines for graph accesses
3. Enabling running different preprocessing routines with a single function call
4. Modular design of classes & files associated with graph algorithms
5. Clear structure of code facilitating manipulation with fine parts such as scheduling policy of single loops

5+ **Set algebra** based modularity for various parts of algorithms

The user can experiment with **algorithmic** ideas (e.g., new algorithms or data structures), **architectural** ideas (e.g., using SIMD or instrinsics), and **design** ideas (e.g., using novel form of load balancing).

GMS software platform & reference implementations

2



GraphMine Suite

Benchmark specification

Details: Section 4 S

Graph **problems** & **algorithms**

- ➜ **Pattern matching** (e.g., clique listing)
- ➜ **Learning** (e.g., link prediction, clustering)
- ➜ **Optimization** (e.g., coloring, minimum cuts)
- ➜ **Reordering** (e.g., degeneracy reordering)

Datasets

- ➜ **Sparse** & **dense**, ➜ many & few **cliques**,
- ➜ High & low **skew** of degree distribution,
- ➜ Many & few **dense** (non-clique) **subgraphs**,
- ➜ different **origins** (purchases, roads, ...)

Reference implementations

Details: Section 5 I

Implementations

- ➜ Algorithms,
- ➜ Optimizations,
- ➜ Preprocessing routines,
- ➜ Load balancing,
- ➜ Graph representations,
- ➜ Data layouts,
- ➜ Graph compression,
- ➜ Parallelizations

Features

➜ Parallel, ➜ Modular, ➜ Scalable, ➜ Fast, ➜ ...

Implemented in

Used by

Benchmarking platform

Details: Sections 3 & 5 P

Features

- ➜ Simple to use,
- ➜ Extensible,
- ➜ Modular,
- ➜ Public.

Key idea for high modularity: use set algebra. Sets and set operations become "modules" that can be implemented in different ways, and still they can be seamlessly combined.

Performance metrics

Details: Sections 5 & 7 M

Traditional

- ➜ Run-time, ➜ Scalability,
- ➜ L3 misses (machine efficiency).

Key idea in a novel metric: count the number of graph patterns mined per second (algorithmic efficiency).

Concurrency analysis

Details: Section 6 C

Aspects

- ➜ Performance (work, depth),
- ➜ Storage, ➜ Tradeoffs.

Platform pipeline stages (toolchain execution) with details on **extensibility** and **modularity**

: a dark background and a cube indicate that a particular part of the design can be substituted by the developer with their own implementation

Load graph into memory

Input graph

1 Build graph representation

Pointers

By default, we use **CSR**

A representation is modular: the user can provide a new representation

Neighborhoods of vertices

2 Define graph accesses

When developing a graph representation, the user also develops the corresponding graph accesses.

Graph accesses

- check d(v)
- iterate over $N(v)$
- check if ∃ (u,v)

3 Apply preprocessing

Example: initial CSR graph representation

Example: reordered CSR (degree order: by neighborhood size)

The user can plug in different preprocessing schemes. We provide a **ready library of reordering schemes**, such as degeneracy or degree reordering (example above).

4 Run graph algorithm

The user can plug in a graph algorithm. We offer >40 reference implementations.

5 Define algorithm building blocks

/* **Example**: Triangle Counting. "tc" is the count of triangles */

The user can plug in variants of fine algorithm blocks such as scheduling policies. GMS facilitates it with appropriate modular implementations

```
tc = 0; init_sets( )
#pragma omp parallel for schedule (...)
for v in V:
  for w in N(v):
    tc += |N(v) ∩ N(w)|
tc /= 3; cleanup( )
```

Most simplicity is enabled by using fine building blocks based on **set algebra** 5+

Gather data

Visualize

How does GMS facilitate extensibility at a given stage?

1 Modular design of classes & files associated with graph representations

2 Well-defined interface (based on set algebra) of routines for graph accesses

3 Enabling running different preprocessing routines with a single function call

4 Modular design of classes & files associated with graph algorithms

5 Clear structure of code facilitating manipulation with fine parts such as scheduling policy of single loops

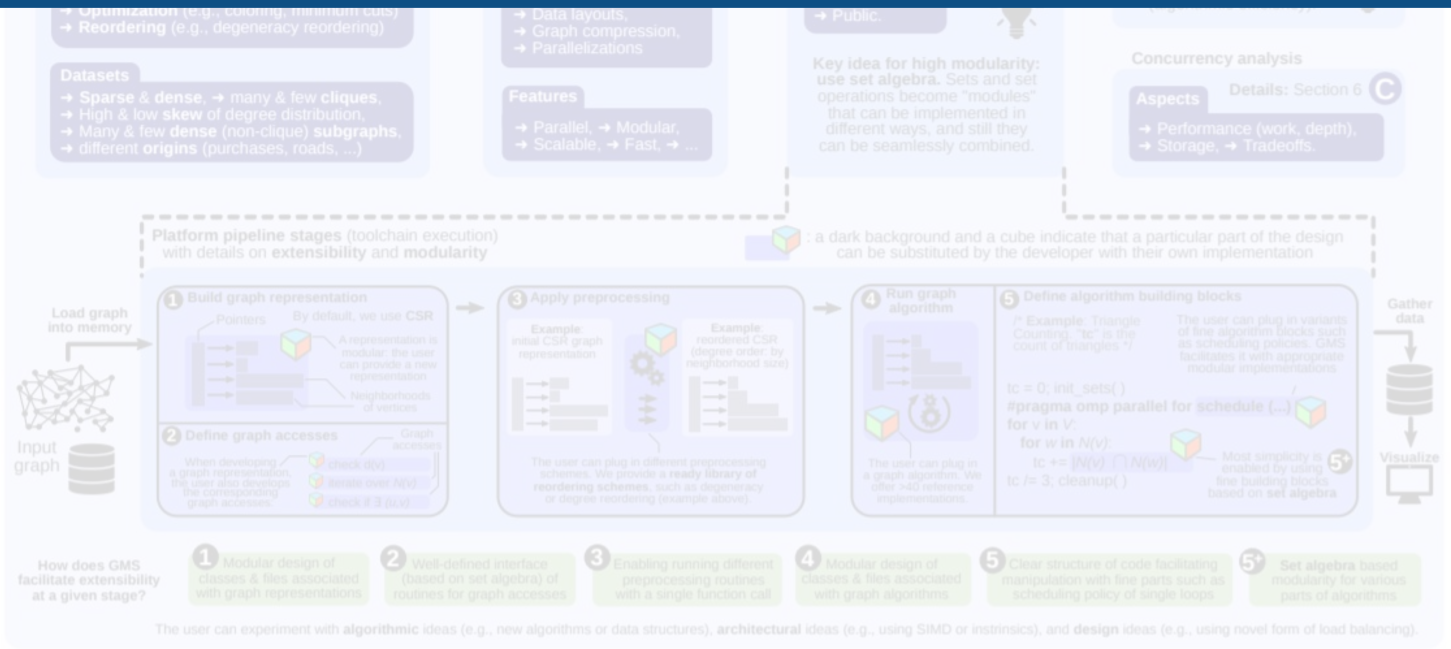
5+ **Set algebra** based modularity for various parts of algorithms

The user can experiment with **algorithmic** ideas (e.g., new algorithms or data structures), **architectural** ideas (e.g., using SIMD or instrinsics), and **design** ideas (e.g., using novel form of load balancing).

GMS software platform & reference implementations

2

Central concept for both programmability and high performance are **set-algebraic formulations**



→ **Optimization** (e.g., coloring, minimum cuts)
→ **Reordering** (e.g., degeneracy reordering)

Datasets

→ **Sparse** & **dense**, → many & few **cliques**,
→ High & low **skew** of degree distribution,
→ Many & few **dense** (non-clique) **subgraphs**,
→ different **origins** (purchases, roads, ...)

→ Data layouts,
→ Graph compression,
→ Parallelizations

Features

→ Parallel, → Modular,
→ Scalable, → Fast, → ...

→ Public.

Key idea for high modularity: use set algebra. Sets and set operations become "modules" that can be implemented in different ways, and still they can be seamlessly combined.

Concurrency analysis

Aspects

Details: Section 6 C

→ Performance (work, depth),
→ Storage, → Tradeoffs.

Platform pipeline stages (toolchain execution) with details on **extensibility** and **modularity**

: a dark background and a cube indicate that a particular part of the design can be substituted by the developer with their own implementation

Load graph into memory

Input graph

1 Build graph representation

Pointers

By default, we use **CSR**

A representation is modular: the user can provide a new representation

Neighborhoods of vertices

2 Define graph accesses

Graph accesses

When developing a graph representation, the user also develops the corresponding graph accesses.

check d(v)
iterate over $N(v)$
check if ∃ (u,v)

3 Apply preprocessing

Example: initial CSR graph representation

Example: reordered CSR (degree order: by neighborhood size)

The user can plug in different preprocessing schemes. We provide a **ready library of reordering schemes**, such as degeneracy or degree reordering (example above).

4 Run graph algorithm

The user can plug in a graph algorithm. We offer >40 reference implementations.

5 Define algorithm building blocks

```
/* Example: Triangle
Counting. "tc" is the
count of triangles */

tc = 0; init_sets( )
#pragma omp parallel for schedule (...)
for v in V:
  for w in N(v):
    tc += |N(v) ∩ N(w)|
tc /= 3; cleanup( )
```

The user can plug in variants of fine algorithm blocks such as scheduling policies. GMS facilitates it with appropriate modular implementations

Most simplicity is enabled by using fine building blocks based on **set algebra** 5+

Gather data

Visualize

How does GMS facilitate extensibility at a given stage?

1 Modular design of classes & files associated with graph representations

2 Well-defined interface (based on set algebra) of routines for graph accesses

3 Enabling running different preprocessing routines with a single function call

4 Modular design of classes & files associated with graph algorithms

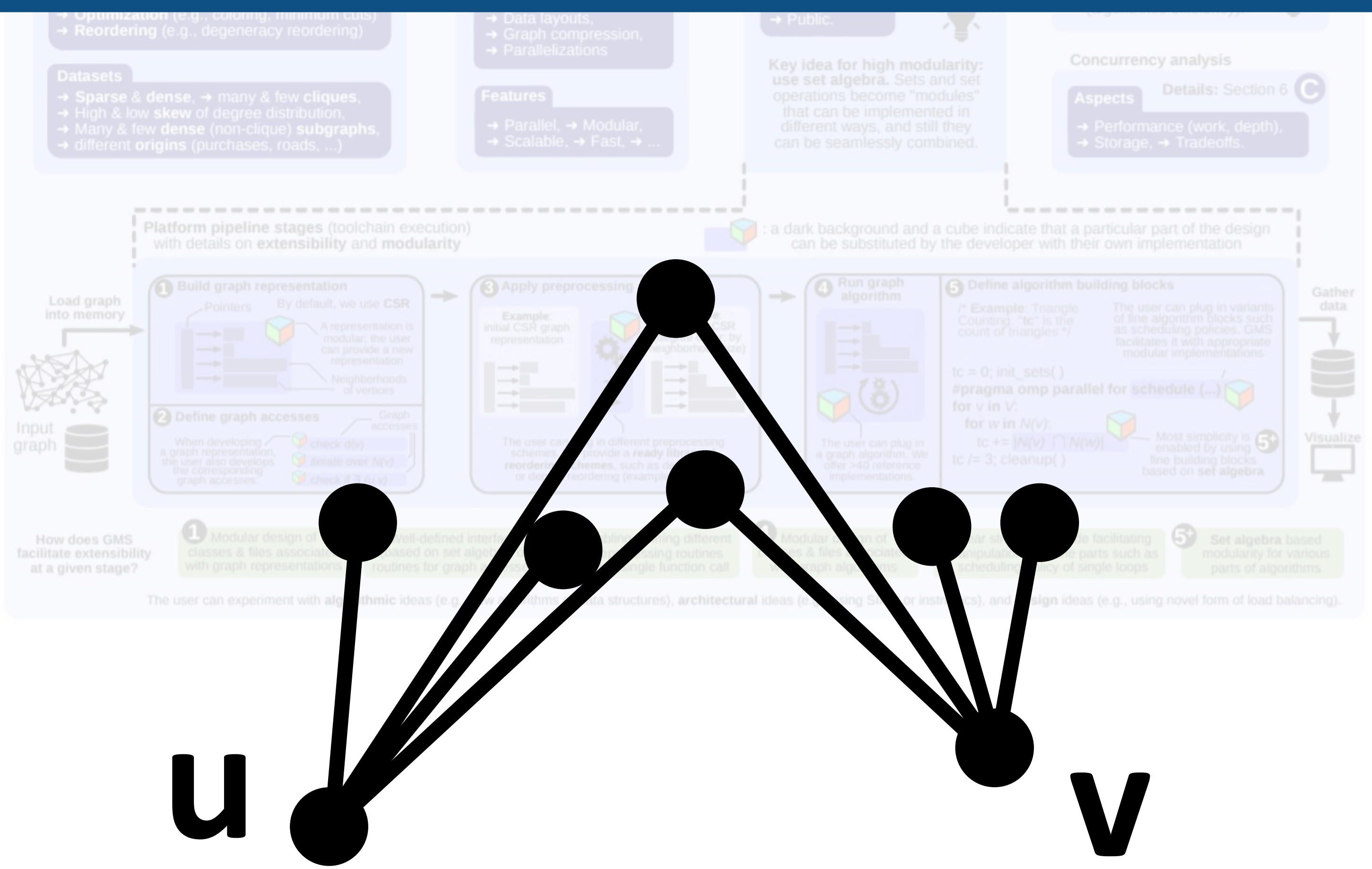
5 Clear structure of code facilitating manipulation with fine parts such as scheduling policy of single loops

5+ **Set algebra** based modularity for various parts of algorithms

The user can experiment with **algorithmic** ideas (e.g., new algorithms or data structures), **architectural** ideas (e.g., using SIMD or instrinsics), and **design** ideas (e.g., using novel form of load balancing).

GMS software platform & reference implementations

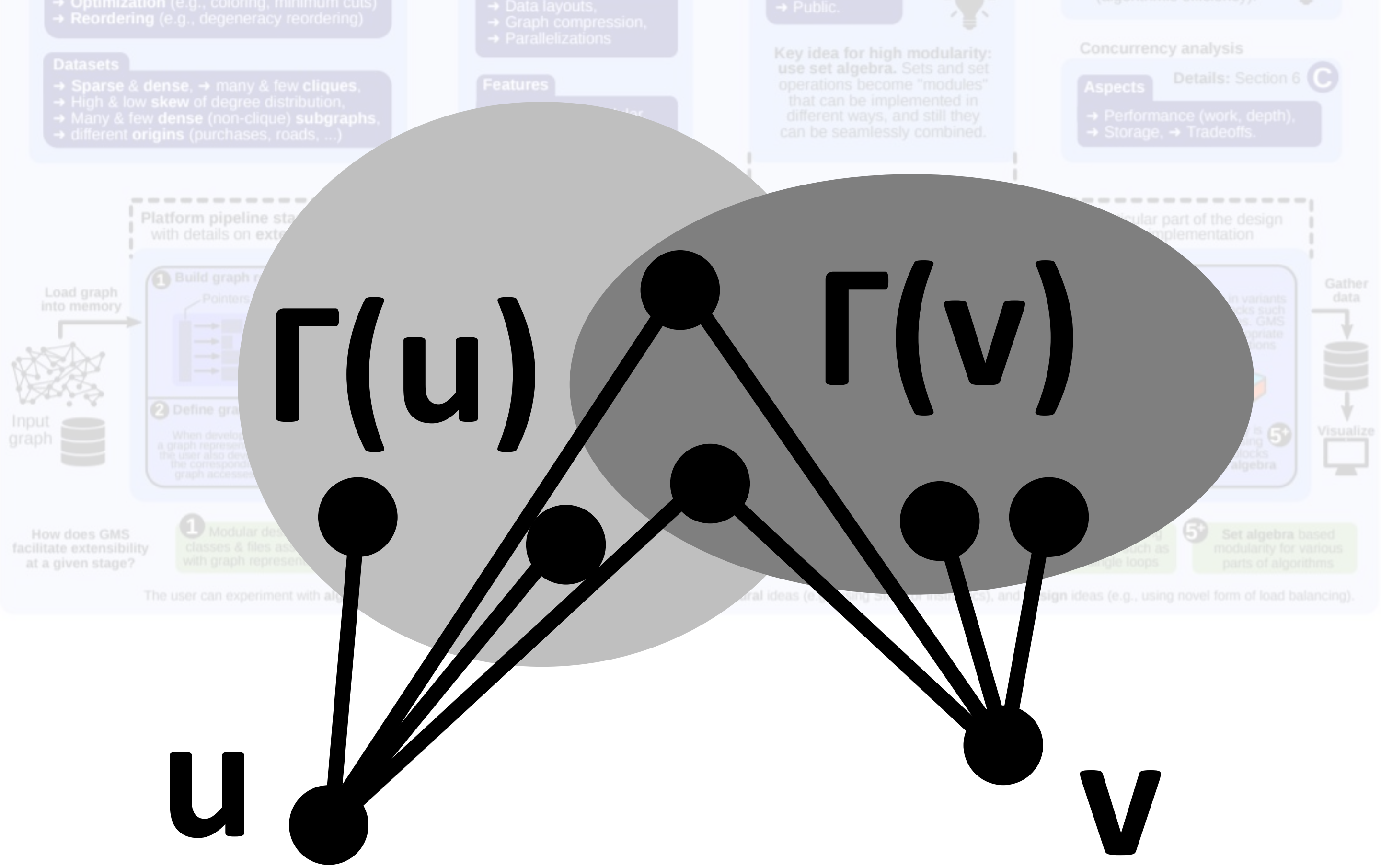
Central concept for both programmability and high performance are **set-algebraic formulations**


→ Optimization (e.g., coloring, minimum cuts)
→ Reordering (e.g., degeneracy reordering)
Datasets
→ Sparse & dense, → many & few cliques,
→ High & low skew of degree distribution,
→ Many & few dense (non-clique) subgraphs,
→ different origins (purchases, roads, ...)
→ Data layouts,
→ Graph compression,
→ Parallelizations
Features
→ Parallel, → Modular,
→ Scalable, → Fast, → ...
→ Public.
Key idea for high modularity: use set algebra. Sets and set operations become "modules" that can be implemented in different ways, and still they can be seamlessly combined.
Concurrency analysis
Aspects
Details: Section 6
→ Performance (work, depth),
→ Storage, → Tradeoffs
Platform pipeline stages (toolchain execution) with details on extensibility and modularity
: a dark background and a cube indicate that a particular part of the design can be substituted by the developer with their own implementation
Load graph into memory
Input graph
1 Build graph representation
Pointers
By default, we use CSR
2 Define graph accesses
3 Apply preprocessing
4 Run graph algorithm
5 Define algorithm building blocks
tc = 0; init_sets( )
#pragma omp parallel for schedule (...)
for v in V:
for w in N(v):
tc += |N(v) ∩ N(w)|
tc /= 3; cleanup( )
Gather data
Visualize
How does GMS facilitate extensibility at a given stage?
5* Set algebra based modularity for various parts of algorithms
u
v

GMS software platform & reference implementations

2

Central concept for both programmability and high performance are **set-algebraic formulations**

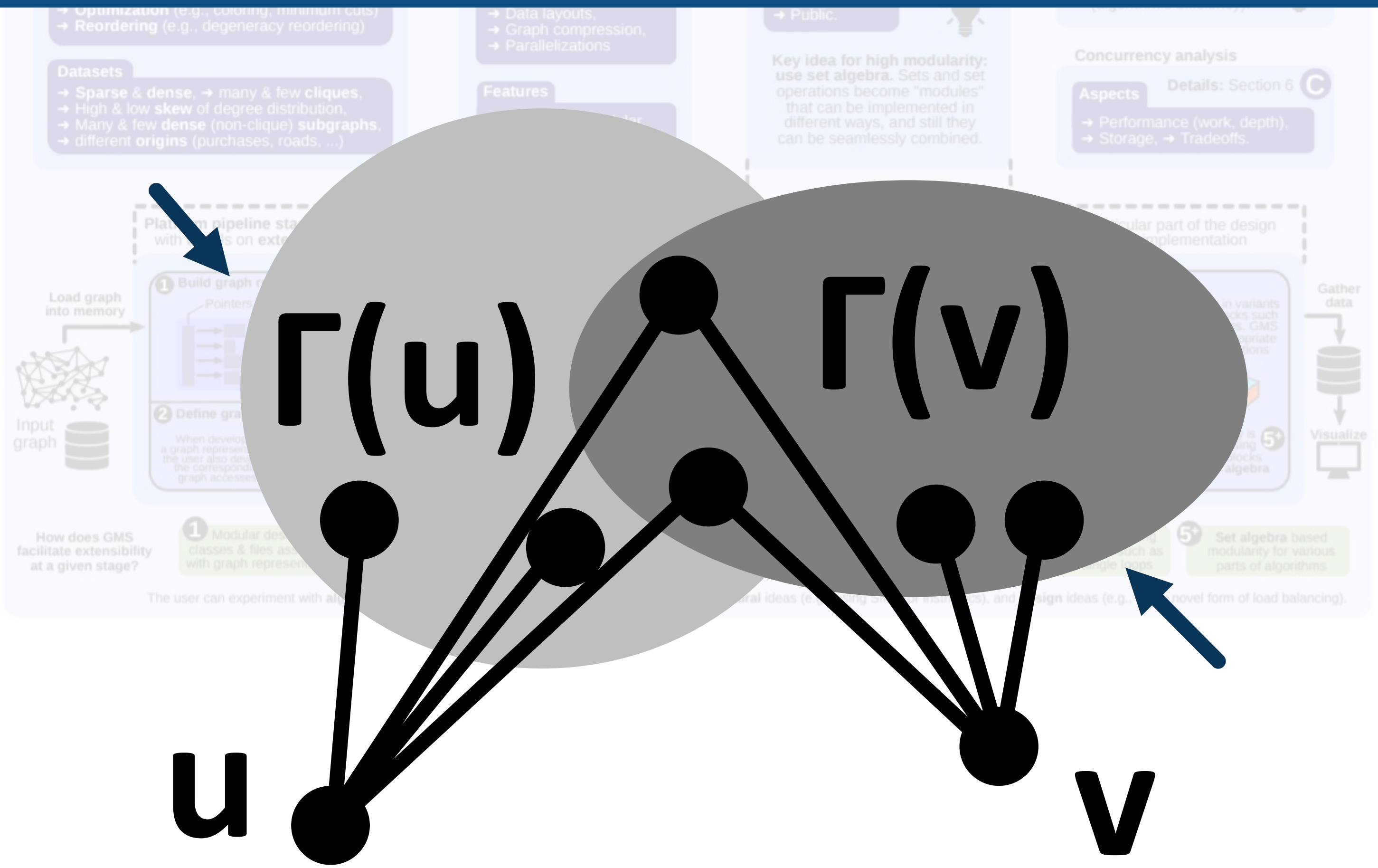


GMS software platform & reference implementations

2

Central concept for both programmability and high performance are **set-algebraic formulations**

Sets

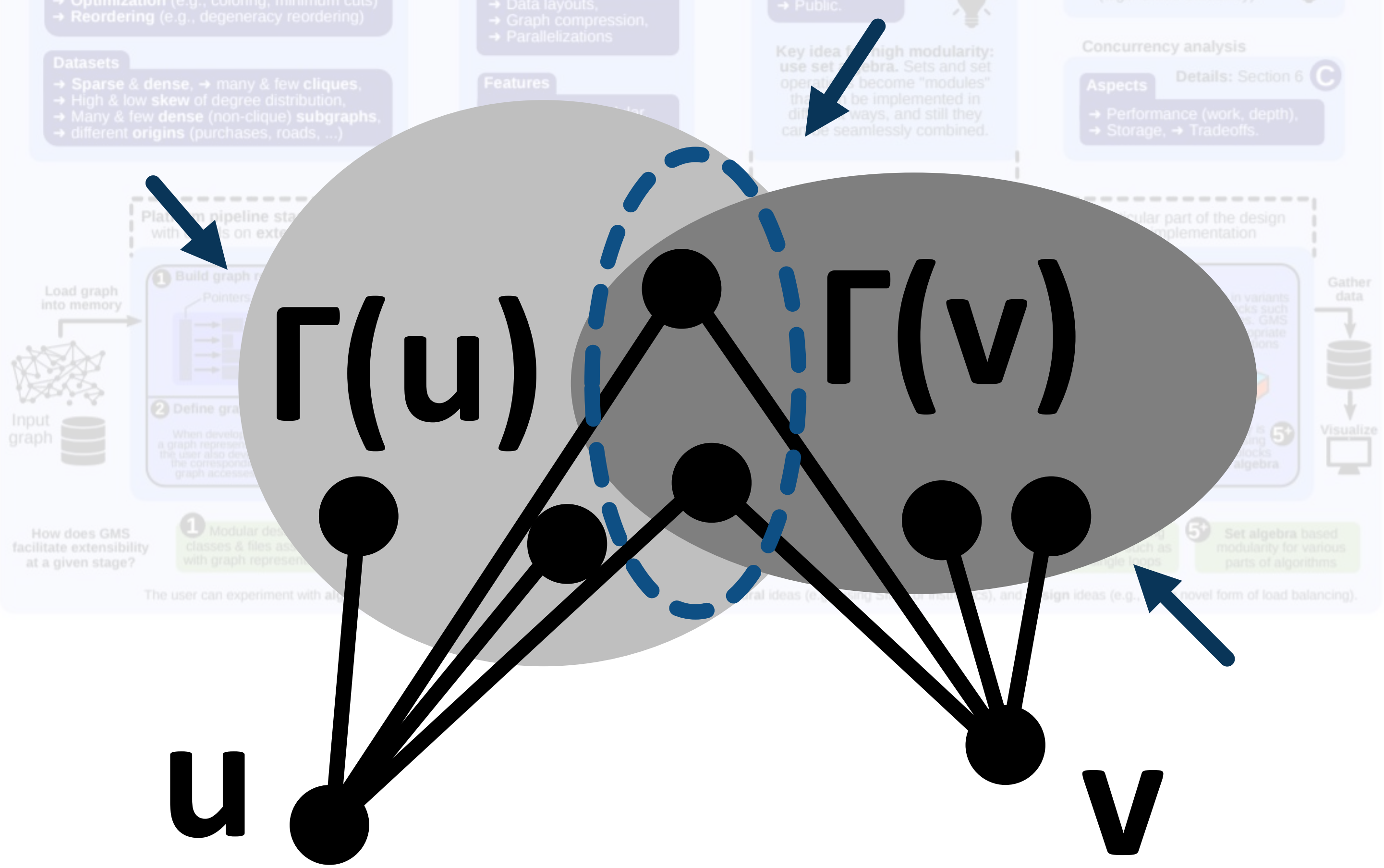


GMS software platform & reference implementations

2

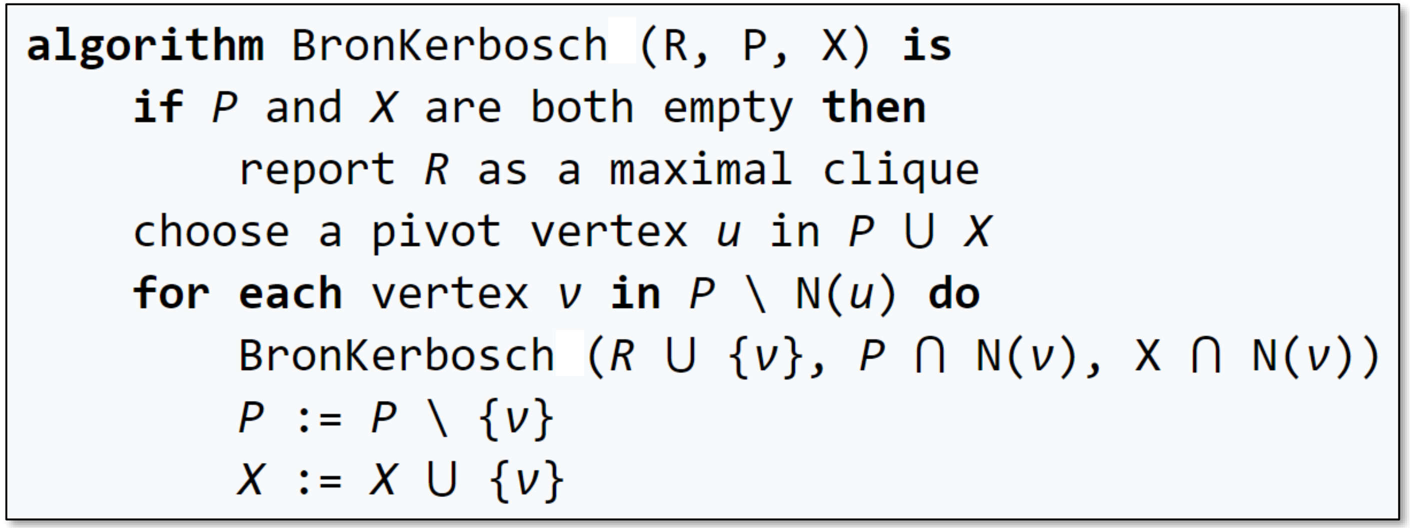
Central concept for both <u>programmability</u> and <u>high performance</u> are <u>**set-algebraic formulations**</u>

Sets



Set operations

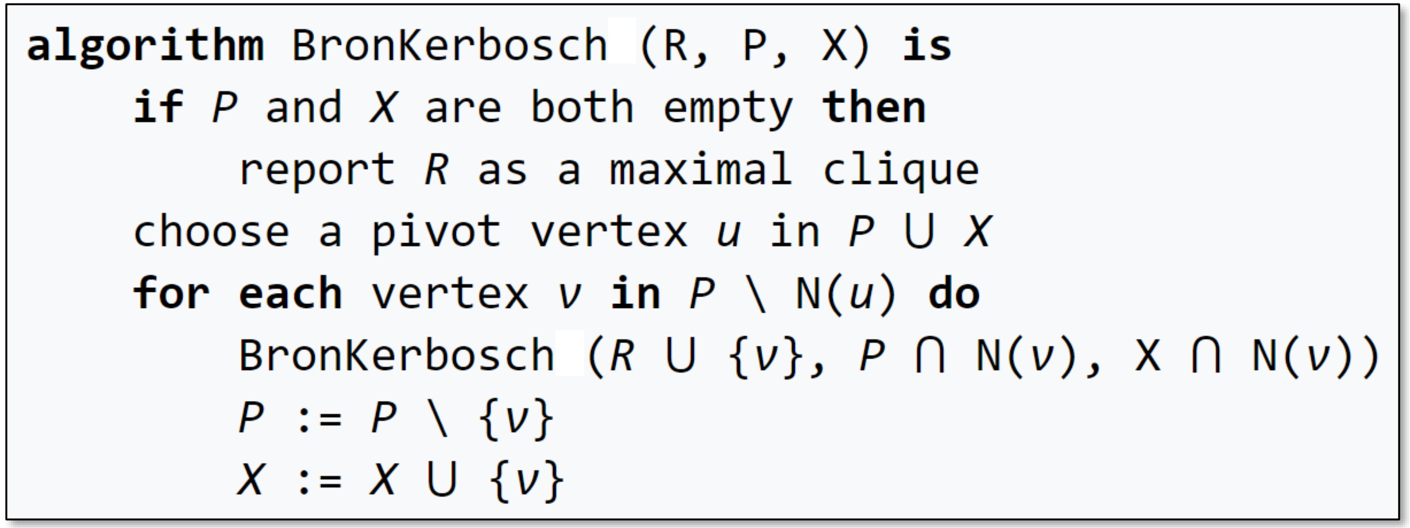
Programmable and High Performance Graph Mining: A Brief Summary



```
algorithm BronKerbosch (R, P, X) is
    if $P$ and $X$ are both empty then
        report $R$ as a maximal clique
    choose a pivot vertex $u$ in $P \cup X$
    for each vertex $v$ in $P \setminus N(u)$ do
        BronKerbosch ($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
        $P := P \setminus \{v\}$
        $X := X \cup \{v\}$
```

Programmable and High Performance Graph Mining: A Brief Summary

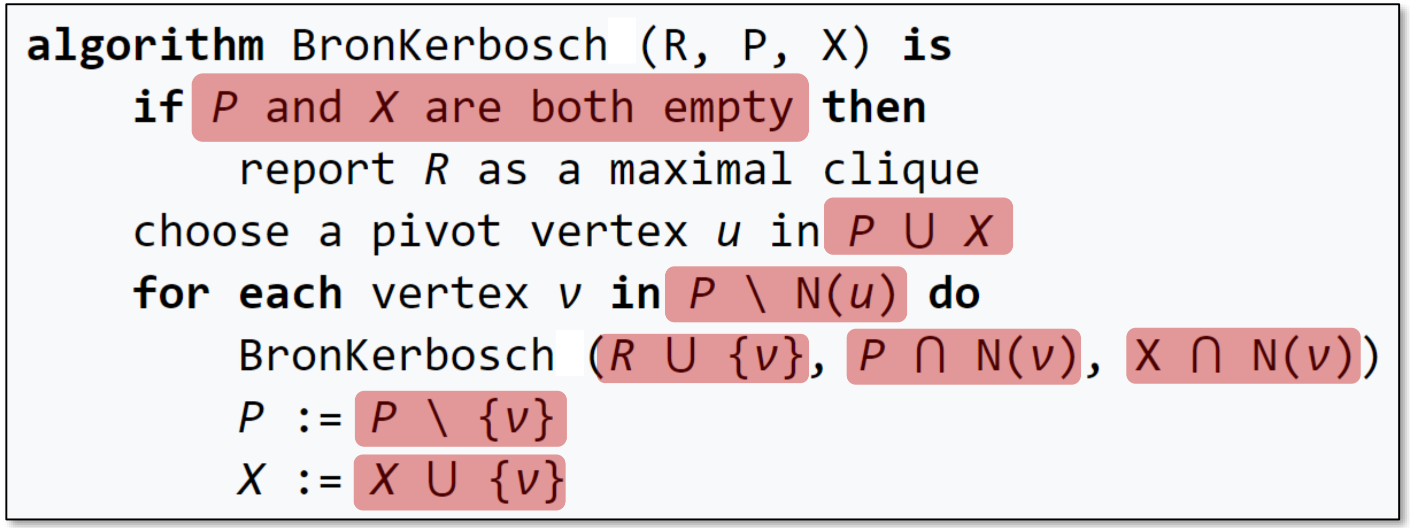
Key idea for both: use **set algebra** building blocks



```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Programmable and High Performance Graph Mining: A Brief Summary

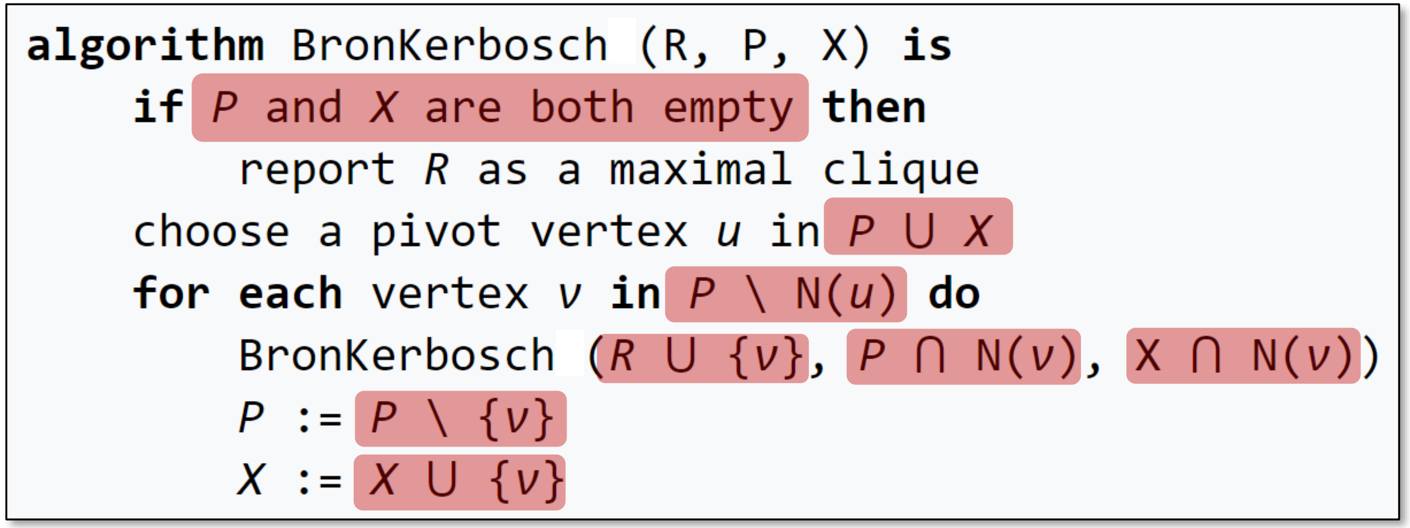
Key idea for both: use **set algebra** building blocks



```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Programmable and High Performance Graph Mining: A Brief Summary

Key idea for both: use **set algebra** building blocks

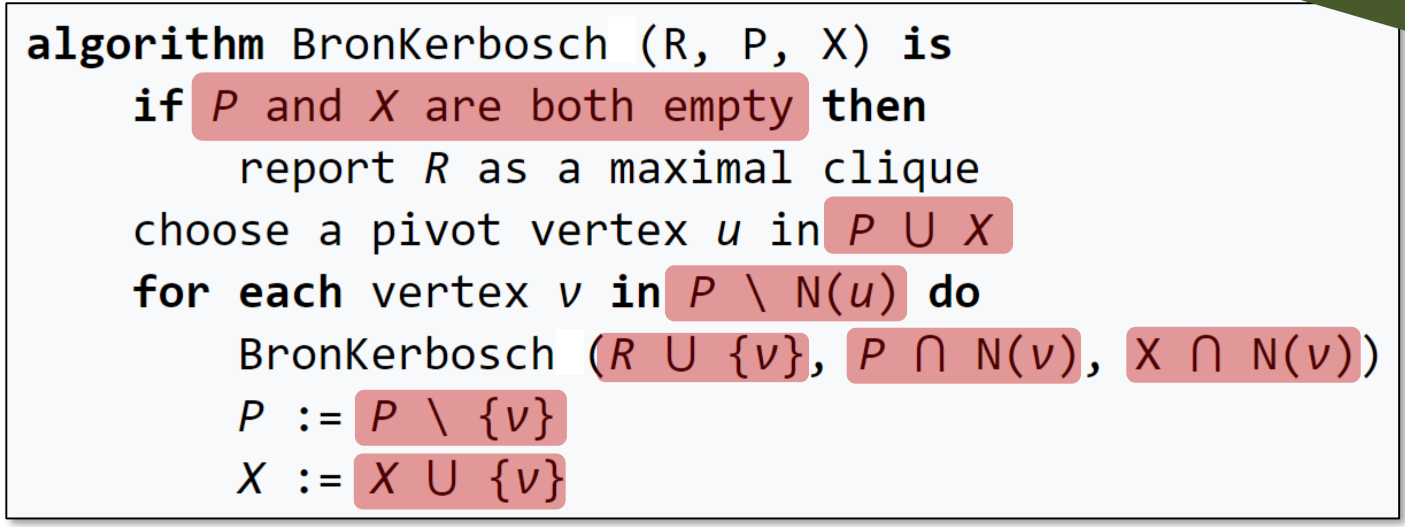


```
algorithm BronKerbosch (R, P, X) is
    if $P$ and $X$ are both empty then
        report $R$ as a maximal clique
    choose a pivot vertex $u$ in $P \cup X$
    for each vertex $v$ in $P \setminus N(u)$ do
        BronKerbosch ($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
        $P := P \setminus \{v\}$
        $X := X \cup \{v\}$
```

Prevalence of set operations in graph mining algorithms & problems

Programmable and High Performance Graph Mining: A Brief Summary

Key idea for both: use **set algebra** building blocks



```
algorithm BronKerbosch (R, P, X) is
    if $P$ and $X$ are both empty then
        report $R$ as a maximal clique
    choose a pivot vertex $u$ in $P \cup X$
    for each vertex $v$ in $P \setminus N(u)$ do
        BronKerbosch ($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
        $P$ := $P \setminus \{v\}$
        $X$ := $X \cup \{v\}$
```

✓ Generality

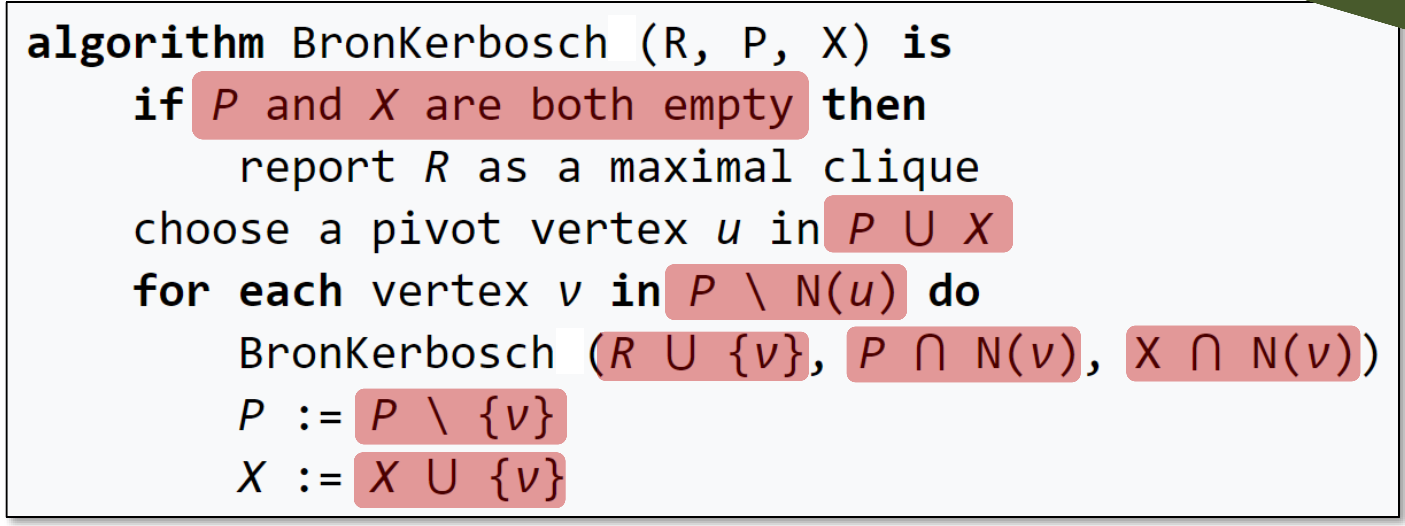
Prevalence of set operations in graph mining algorithms & problems

Programmable and High Performance Graph Mining: A Brief Summary

Key idea for both: use **set algebra** building blocks

Generality

Prevalence of set operations in graph mining algorithms & problems



```
algorithm BronKerbosch (R, P, X) is
    if $P$ and $X$ are both empty then
        report $R$ as a maximal clique
    choose a pivot vertex $u$ in $P \cup X$
    for each vertex $v$ in $P \setminus N(u)$ do
        BronKerbosch ($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
        $P := P \setminus \{v\}$
        $X := X \cup \{v\}$
```

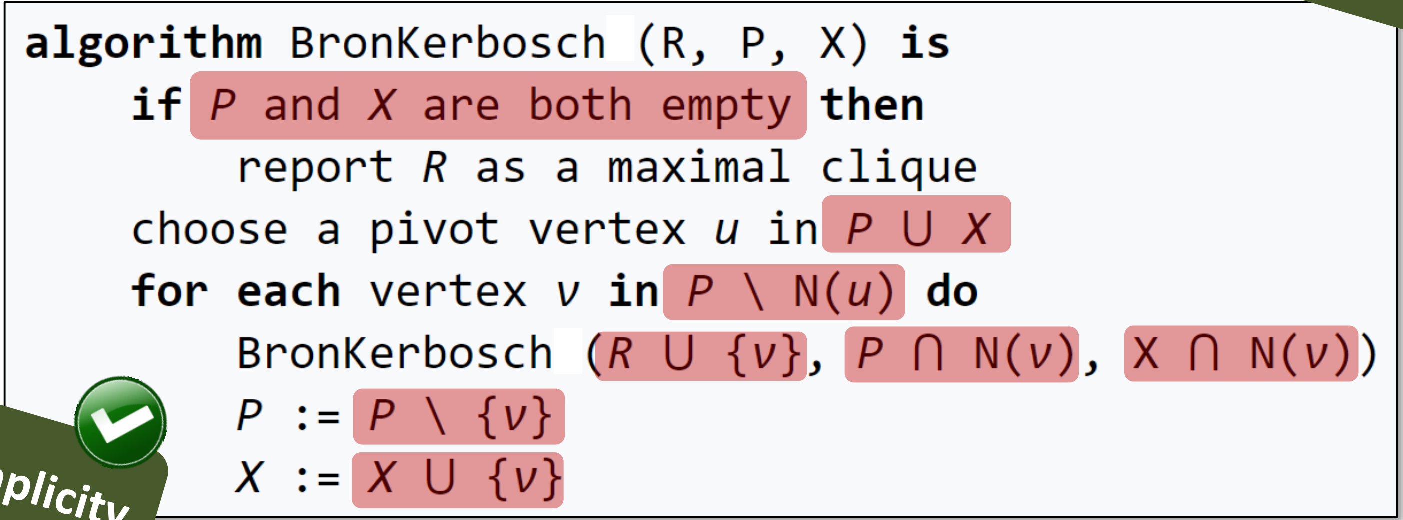
Breaking down complex graph mining algorithms **into simple building blocks**, which can be separately optimized and coded

Programmable and High Performance Graph Mining: A Brief Summary

Key idea for both: use **set algebra** building blocks

Generality

Prevalence of set operations in graph mining algorithms & problems



```
algorithm BronKerbosch (R, P, X) is
    if $P$ and $X$ are both empty then
        report $R$ as a maximal clique
    choose a pivot vertex $u$ in $P \cup X$
    for each vertex $v$ in $P \setminus N(u)$ do
        BronKerbosch ($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
        $P$ := $P \setminus \{v\}$
        $X$ := $X \cup \{v\}$
```

Simplicity

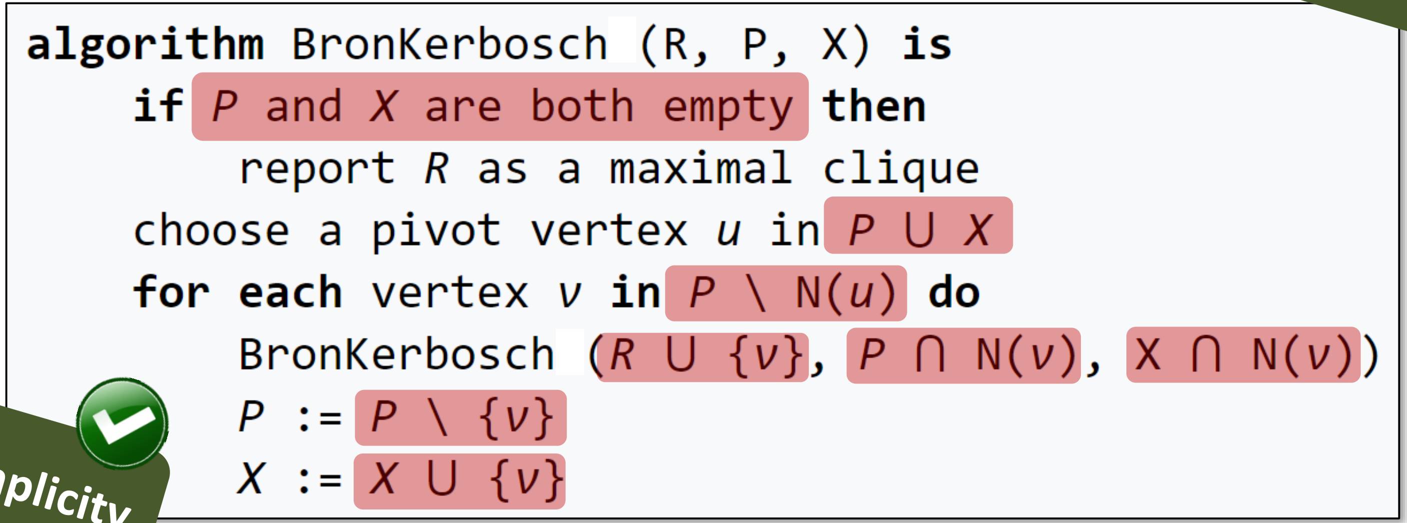
Breaking down complex graph mining algorithms **into simple building blocks**, which can be separately optimized and coded

Programmable and High Performance Graph Mining: A Brief Summary

Key idea for both: use **set algebra** building blocks

Generality

Prevalence of set operations in graph mining algorithms & problems



```
algorithm BronKerbosch (R, P, X) is
    if $P$ and $X$ are both empty then
        report $R$ as a maximal clique
    choose a pivot vertex $u$ in $P \cup X$
    for each vertex $v$ in $P \setminus N(u)$ do
        BronKerbosch ($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
        $P := P \setminus \{v\}$
        $X := X \cup \{v\}$
```

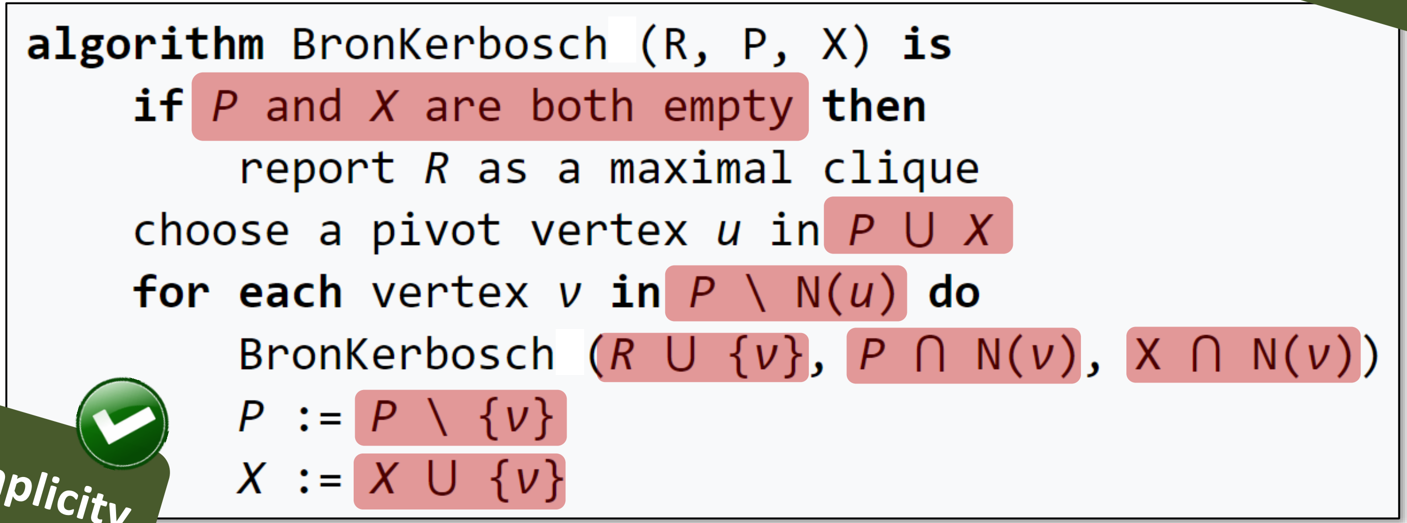
Simplicity

Parallelism across and within **set operations**

Breaking down complex graph mining algorithms **into simple building blocks**, which can be separately optimized and coded

Programmable and High Performance Graph Mining: A Brief Summary

Key idea for both: use **set algebra** building blocks



```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

✓ Generality

✓ Simplicity

Breaking down complex graph mining algorithms **into simple building blocks**, which can be separately optimized and coded

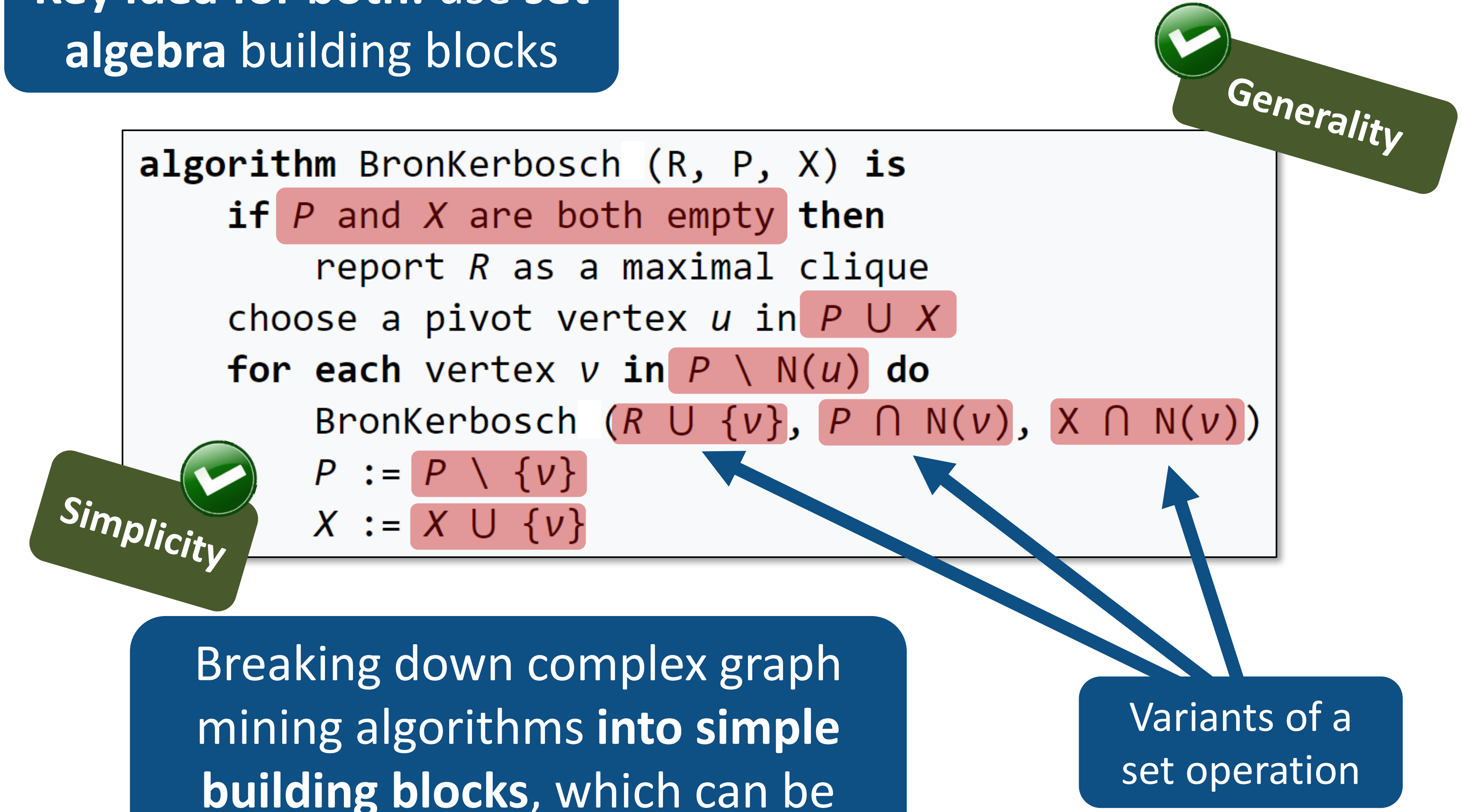
Prevalence of set operations in graph mining algorithms & problems

Parallelism across and within **set operations**

Facilitates **prototyping** and **optimization**

Programmable and High Performance Graph Mining: A Brief Summary

Key idea for both: use **set algebra** building blocks



```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Variants of a set operation

Prevalence of set operations in graph mining algorithms & problems

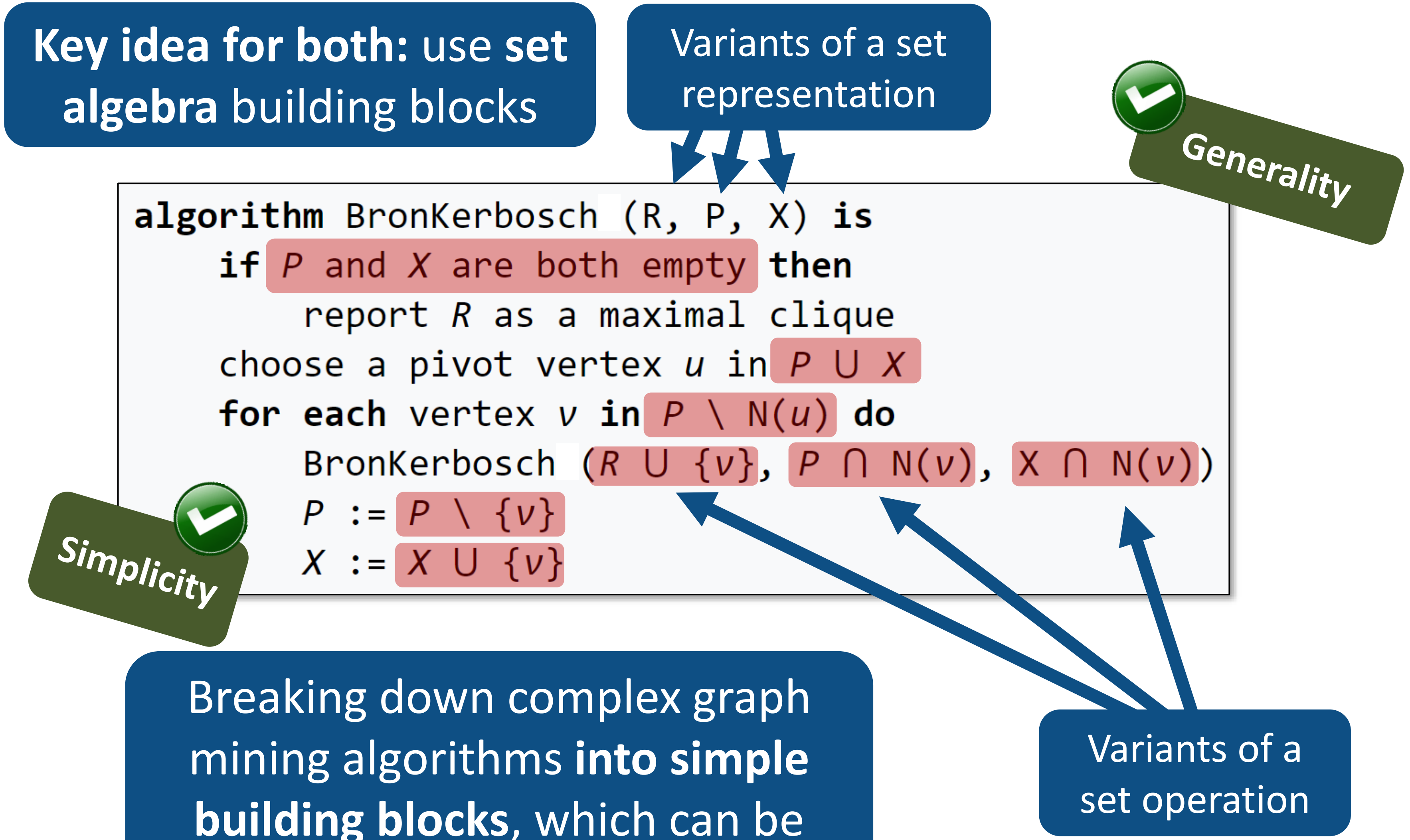
Parallelism across and within **set operations**

Breaking down complex graph mining algorithms **into simple building blocks**, which can be separately optimized and coded

Facilitates **prototyping** and **optimization**

Programmable and High Performance Graph Mining: A Brief Summary

Key idea for both: use **set algebra** building blocks



Variants of a set representation

Generality

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

Simplicity

Variants of a set operation

Prevalence of set operations in graph mining algorithms & problems

Parallelism across and within **set operations**

Breaking down complex graph mining algorithms **into simple building blocks**, which can be separately optimized and coded

Facilitates **prototyping** and **optimization**

Programmable and High Performance Graph Mining: A Brief Summary

Key idea for both: use **set algebra** building blocks

Variants of a set representation

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ⋃ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

✓ Generality

✓ High performance

✓ Simplicity

Prevalence of set operations in graph mining algorithms & problems

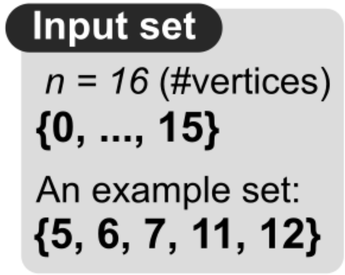
Parallelism across and within **set operations**

Breaking down complex graph mining algorithms **into simple building blocks**, which can be separately optimized and coded

Variants of a set operation

Facilitates **prototyping** and **optimization**

Example Advantages of Set Algebra Building Blocks



Input set

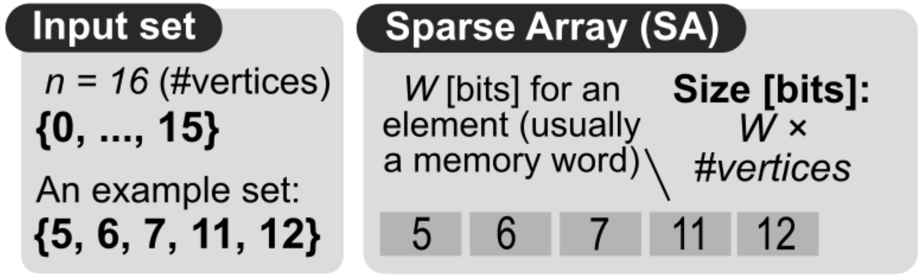
n = *16* (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Example Advantages of Set Algebra Building Blocks



Input set

n = 16 (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

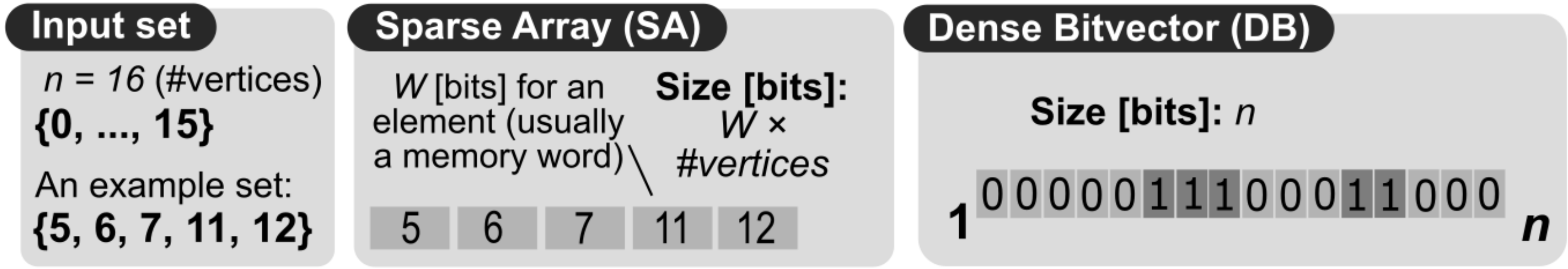
Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

5	6	7	11	12

Example Advantages of Set Algebra Building Blocks



Input set

n = 16 (#vertices)
{0, ..., 15}

An example set:
{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

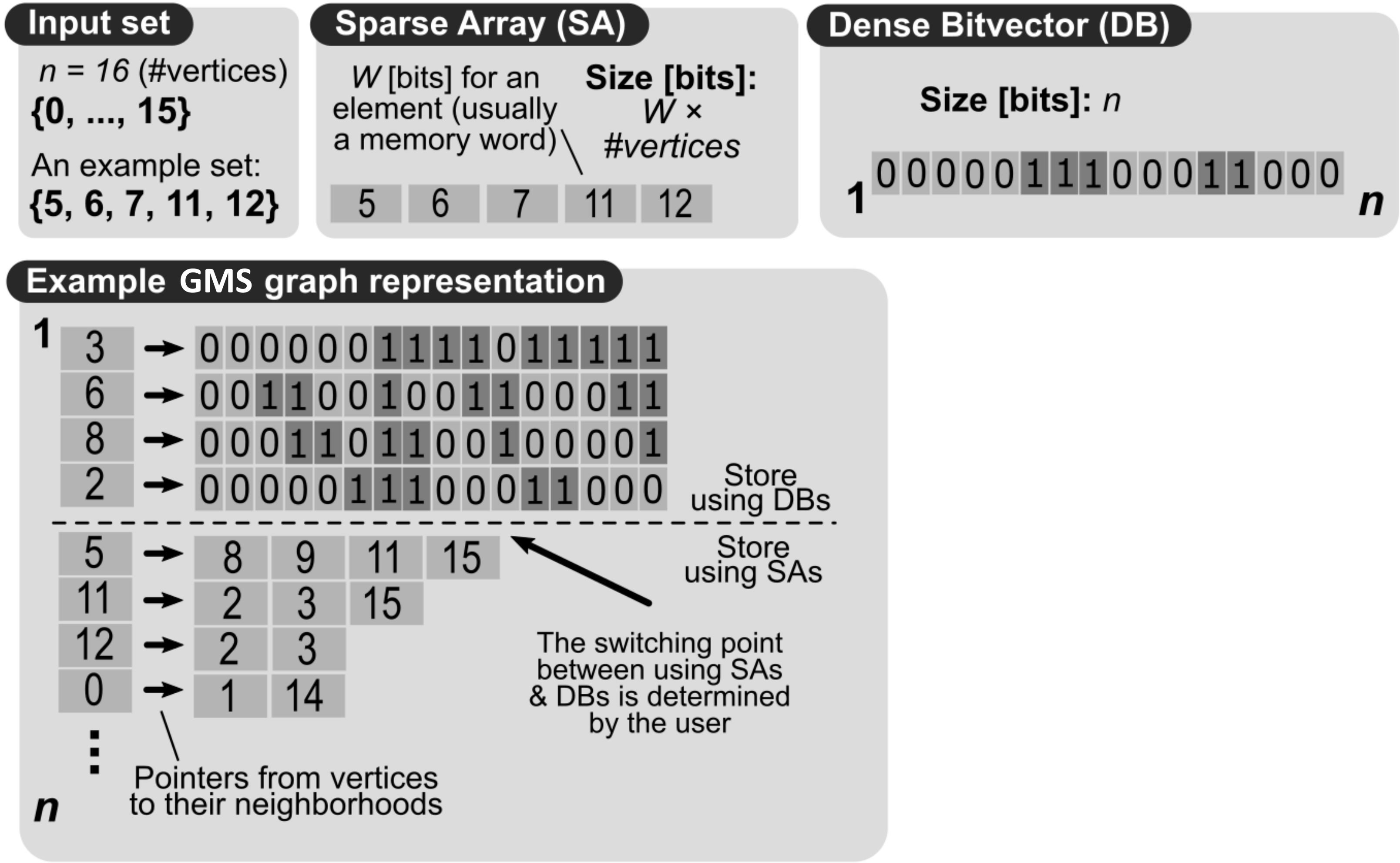
5 6 7 11 12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 ***n***

Example Advantages of Set Algebra Building Blocks



Input set

n = 16 (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

5	6	7	11	12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*

Example GMS graph representation

```
1  3 → 0 0 0 0 0 0 1 1 1 1 0 1 1 1 1 1
   6 → 0 0 1 1 0 0 1 0 0 1 1 0 0 0 1 1
   8 → 0 0 0 1 1 0 1 1 0 0 1 0 0 0 0 1
   2 → 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0
-----------------------------------------
   5 → 8 9 11 15
  11 → 2 3 15
  12 → 2 3
   0 → 1 14
   ⋮
n
```

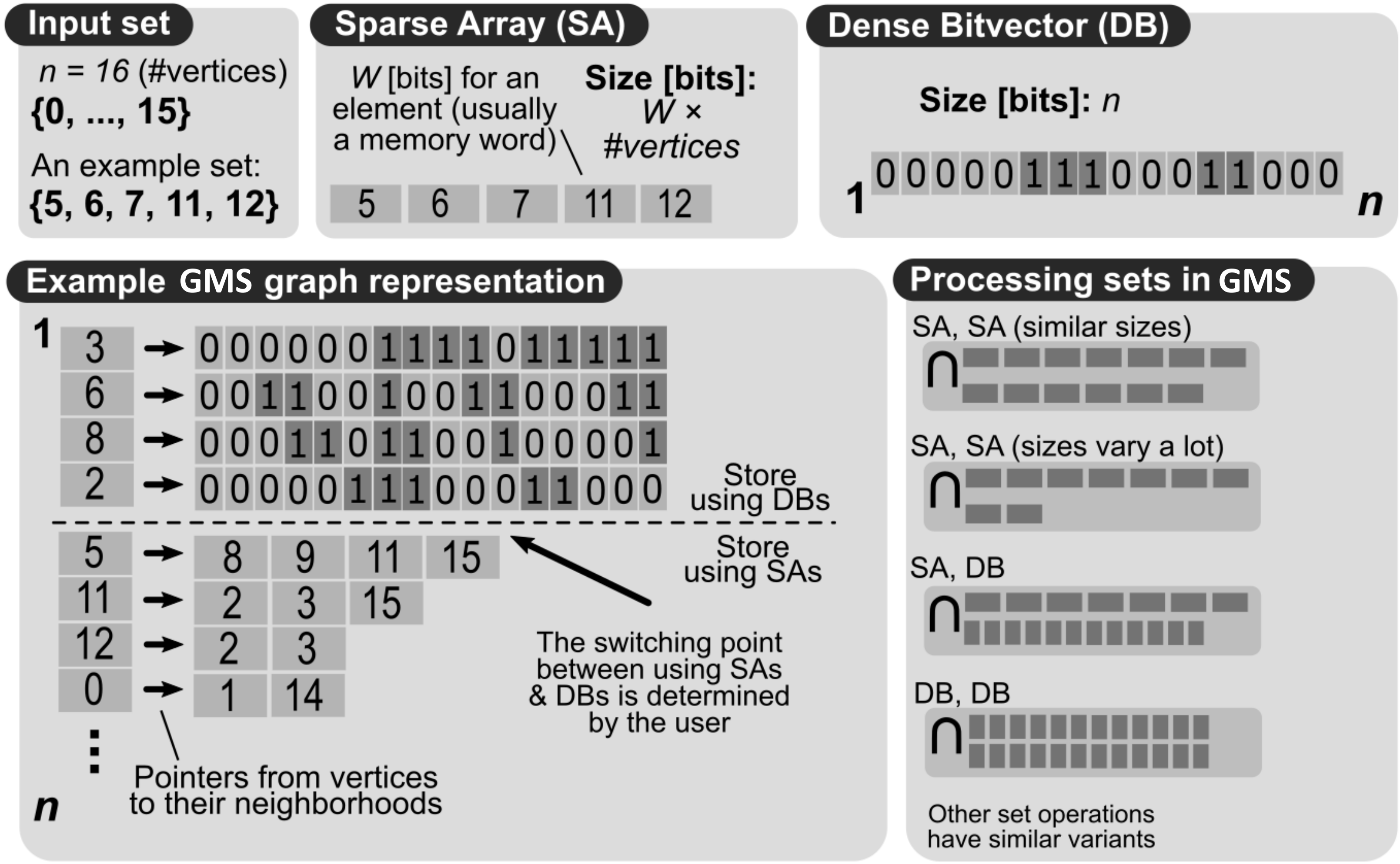
Store using DBs

Store using SAs

The switching point between using SAs & DBs is determined by the user

Pointers from vertices to their neighborhoods

Example Advantages of Set Algebra Building Blocks



Input set

n = 16 (#vertices)
{0, ..., 15}

An example set:
{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

5 6 7 11 12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*

Example GMS graph representation

1
3 → 0 0 0 0 0 0 1 1 1 1 0 1 1 1 1 1
6 → 0 0 1 1 0 0 1 0 0 1 1 0 0 0 1 1
8 → 0 0 0 1 1 0 1 1 0 0 1 0 0 0 0 1
2 → 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0

Store using DBs

5 → 8 9 11 15
11 → 2 3 15
12 → 2 3
0 → 1 14
⋮
n

Store using SAs

The switching point between using SAs & DBs is determined by the user

Pointers from vertices to their neighborhoods

Processing sets in GMS

SA, SA (similar sizes)

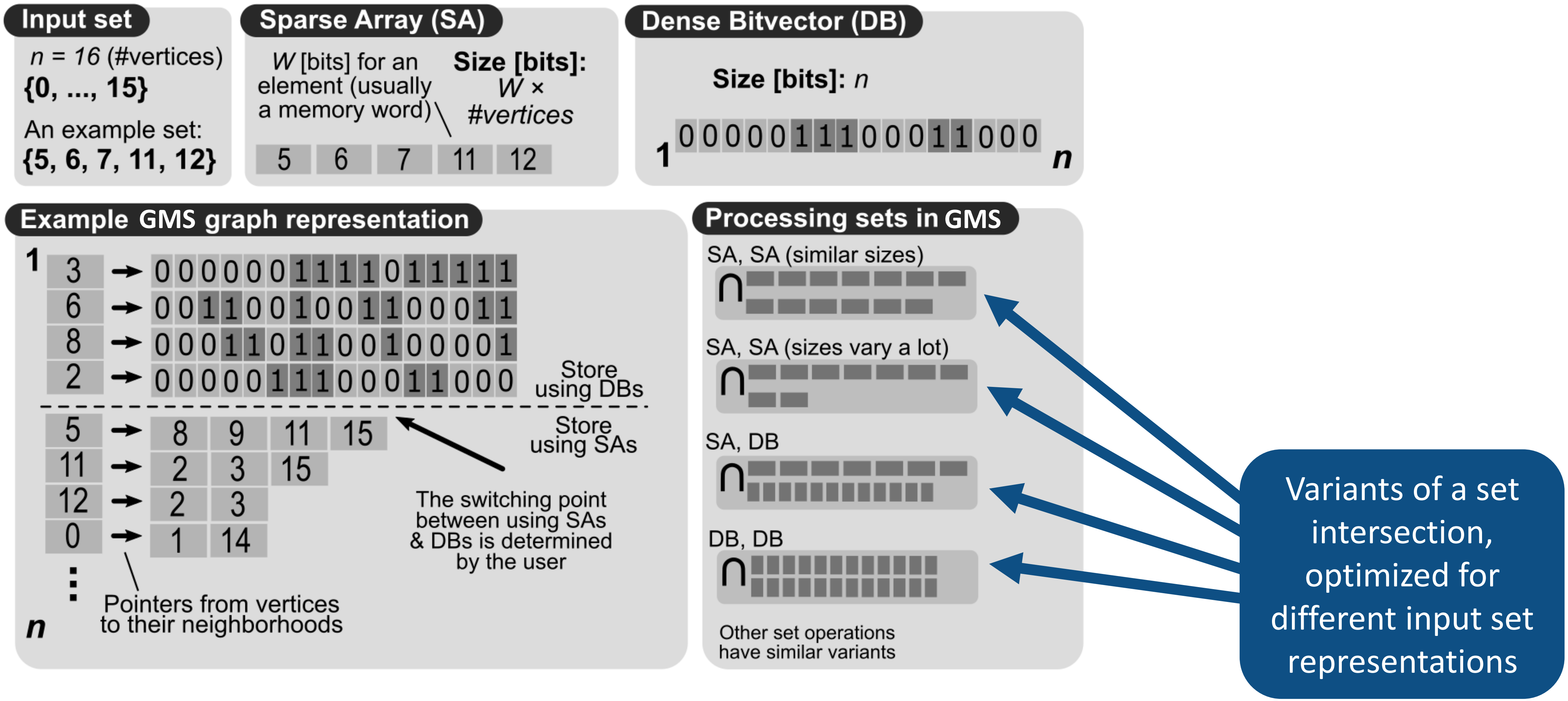
SA, SA (sizes vary a lot)

SA, DB

DB, DB

Other set operations have similar variants

Example Advantages of Set Algebra Building Blocks



Input set

n = 16 (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

5 6 7 11 12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*

Example GMS graph representation

1

3 → 0 0 0 0 0 0 1 1 1 1 0 1 1 1 1 1

6 → 0 0 1 1 0 0 1 0 0 1 1 0 0 0 1 1

8 → 0 0 0 1 1 0 1 1 0 0 1 0 0 0 0 1

2 → 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0

Store using DBs

5 → 8 9 11 15

11 → 2 3 15

12 → 2 3

0 → 1 14

⋮

n

Store using SAs

The switching point between using SAs & DBs is determined by the user

Pointers from vertices to their neighborhoods

Processing sets in GMS

SA, SA (similar sizes)

∩

SA, SA (sizes vary a lot)

∩

SA, DB

∩

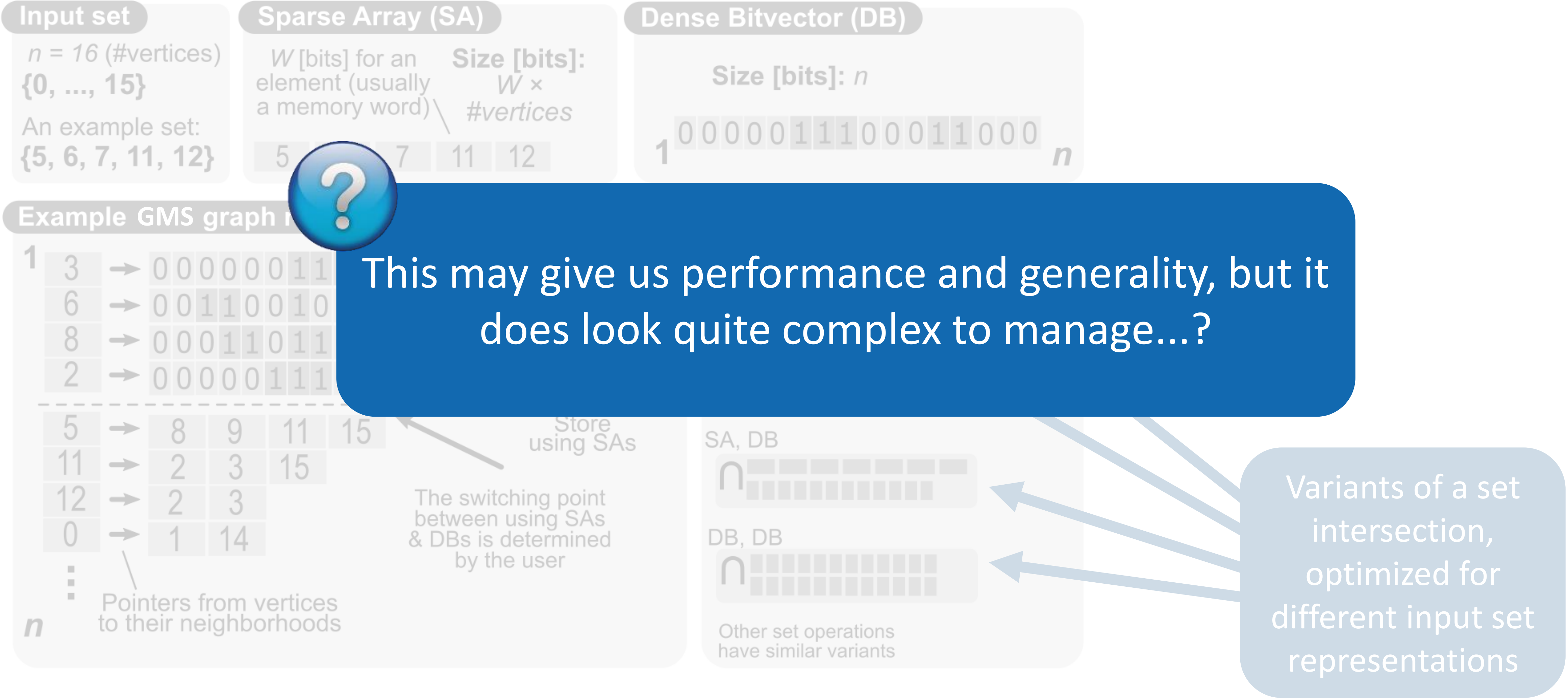
DB, DB

∩

Other set operations have similar variants

Variants of a set intersection, optimized for different input set representations

Example Advantages of Set Algebra Building Blocks



Input set

n = 16 (#vertices)

{0, ..., 15}

An example set:

{5, 6, 7, 11, 12}

Sparse Array (SA)

W [bits] for an element (usually a memory word)

Size [bits]: *W* × *#vertices*

5 7 11 12

Dense Bitvector (DB)

Size [bits]: *n*

1 0 0 0 0 0 1 1 1 0 0 0 1 1 0 0 0 *n*

Example GMS graph r

1

3 → 0 0 0 0 0 0 1 1

6 → 0 0 1 1 0 0 1 0

8 → 0 0 0 1 1 0 1 1

2 → 0 0 0 0 0 1 1 1

5 → 8 9 11 15

11 → 2 3 15

12 → 2 3

0 → 1 14

⋮

n

Pointers from vertices to their neighborhoods

Store using SAs

The switching point between using SAs & DBs is determined by the user

SA, DB

∩

DB, DB

∩

Other set operations have similar variants

This may give us performance and generality, but it does look quite complex to manage...?

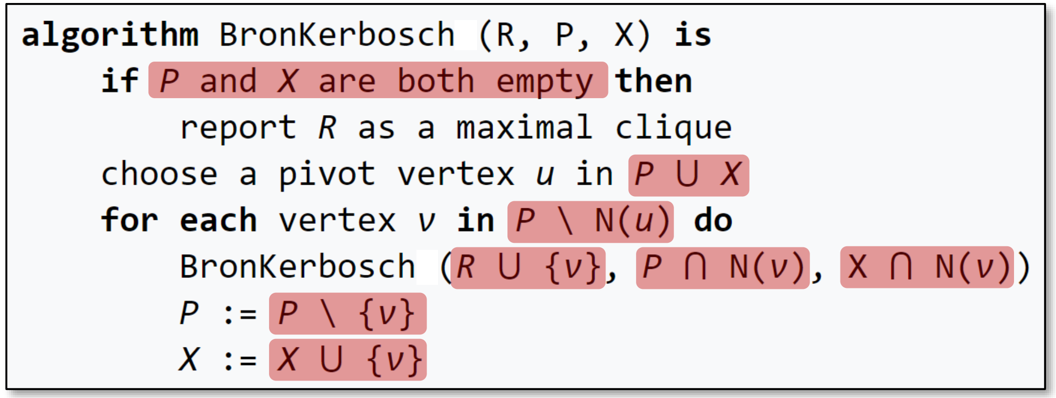
Variants of a set intersection, optimized for different input set representations

Example Advantages of Set Algebra Building Blocks

...It’s all about abstracting away the details with set-centric formulations + modularity

Example Advantages of Set Algebra Building Blocks

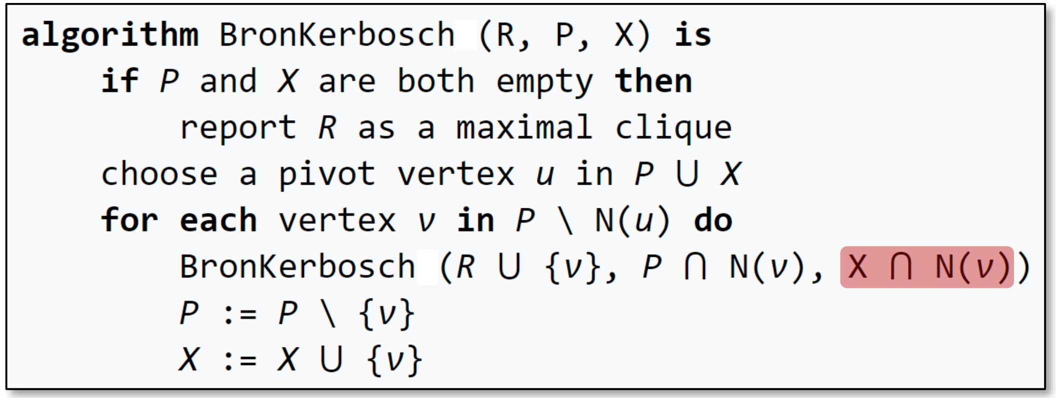
...It's all about abstracting away the details with set-centric formulations + modularity



```
algorithm BronKerbosch (R, P, X) is
    if $P$ and $X$ are both empty then
        report $R$ as a maximal clique
    choose a pivot vertex $u$ in $P \cup X$
    for each vertex $v$ in $P \setminus N(u)$ do
        BronKerbosch ($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
        $P$ := $P \setminus \{v\}$
        $X$ := $X \cup \{v\}$
```

Example Advantages of Set Algebra Building Blocks

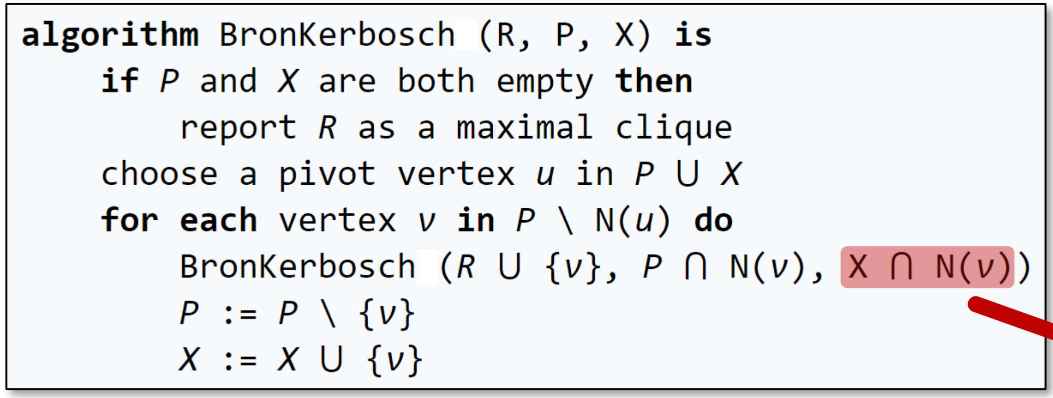
...It’s all about abstracting away the details with set-centric formulations + modularity



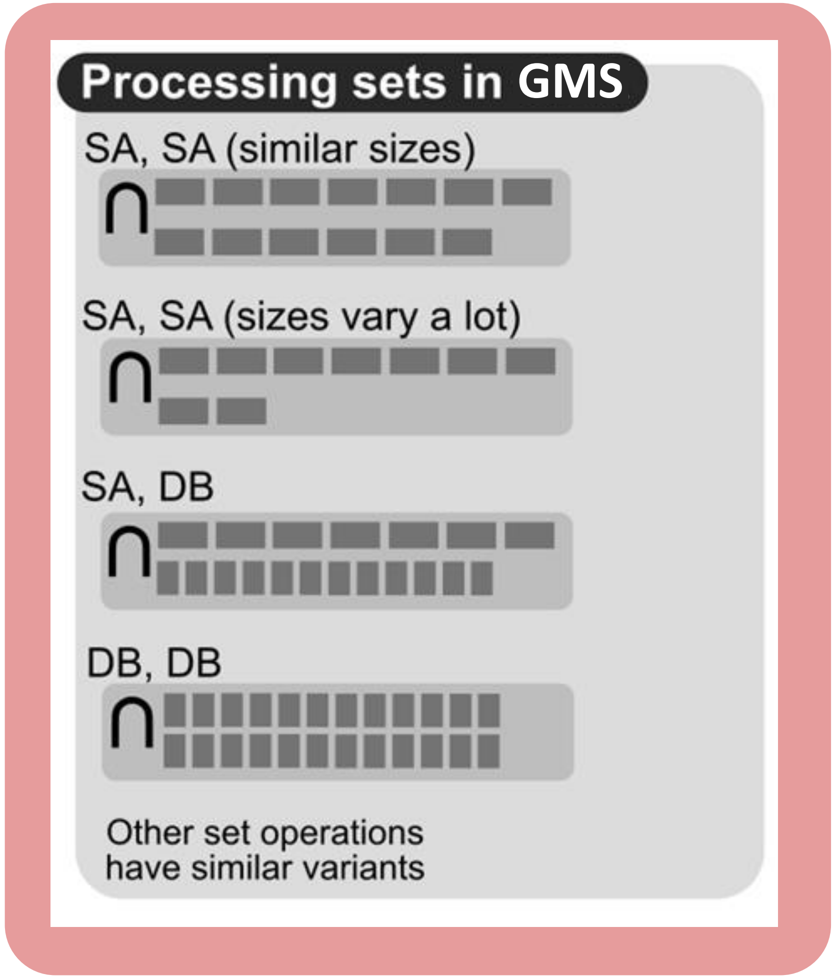
```
algorithm BronKerbosch (R, P, X) is
    if $P$ and $X$ are both empty then
        report $R$ as a maximal clique
    choose a pivot vertex $u$ in $P \cup X$
    for each vertex $v$ in $P \setminus N(u)$ do
        BronKerbosch ($R \cup \{v\}$, $P \cap N(v)$, $X \cap N(v)$)
        $P := P \setminus \{v\}$
        $X := X \cup \{v\}$
```

Example Advantages of Set Algebra Building Blocks

...It’s all about abstracting away the details with set-centric formulations + modularity



```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```



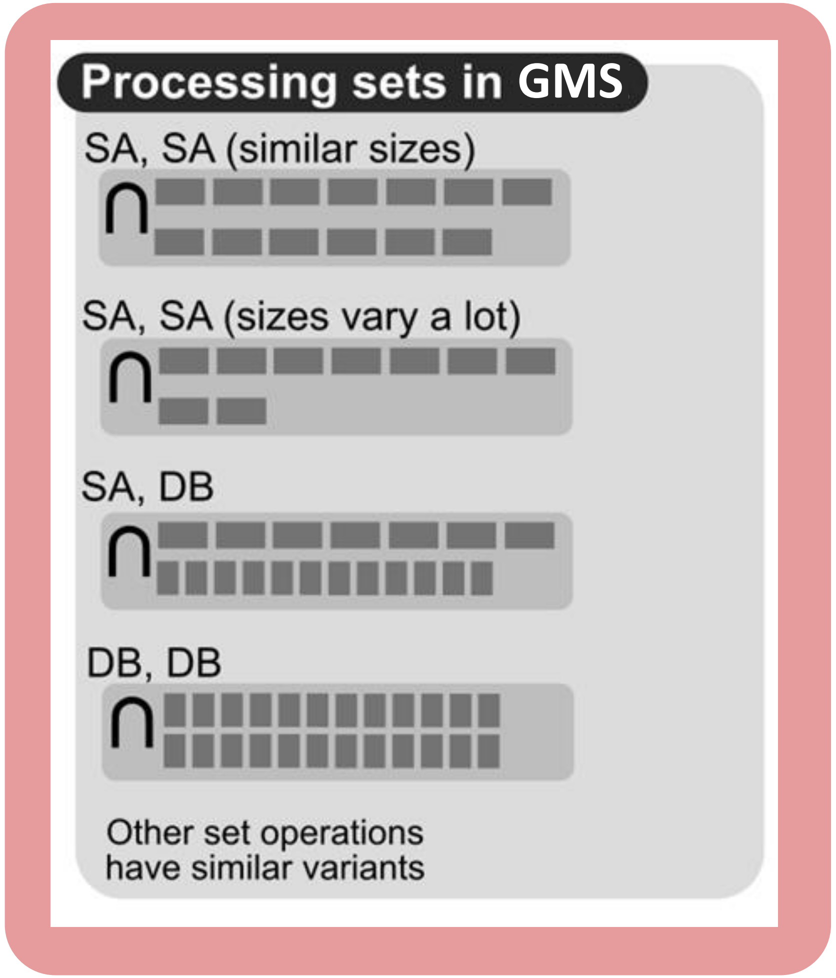
Other set operations have similar variants

Example Advantages of Set Algebra Building Blocks

...It’s all about abstracting away the details with set-centric formulations + modularity

```
algorithm BronKerbosch (R, P, X) is
    if P and X are both empty then
        report R as a maximal clique
    choose a pivot vertex u in P ∪ X
    for each vertex v in P \ N(u) do
        BronKerbosch (R ∪ {v}, P ∩ N(v), X ∩ N(v))
        P := P \ {v}
        X := X ∪ {v}
```

A set-centric formulation of a graph mining algorithm remains simple

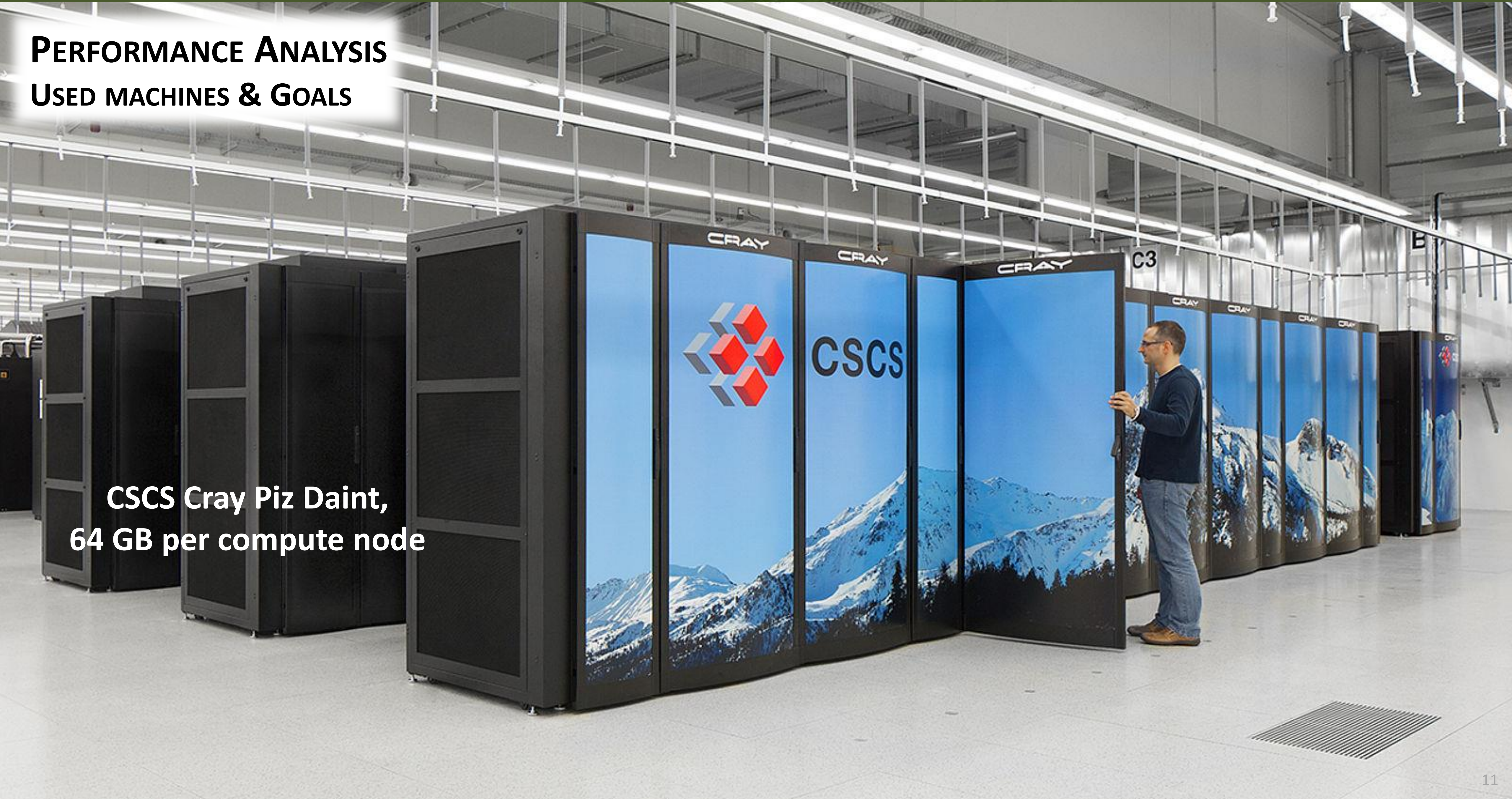

Processing sets in GMS
SA, SA (similar sizes)
∩
SA, SA (sizes vary a lot)
∩
SA, DB
∩
DB, DB
∩
Other set operations have similar variants

Performance Analysis

Used machines & Goals

Performance Analysis

Used machines & Goals



CSCS Cray Piz Daint,
64 GB per compute node

Performance Analysis

Used machines & Goals

CSCS Cray Piz Daint,
64 GB per compute node

CSCS Ault server, 768 GB of DRAM

Performance Analysis
Used machines & Goals

Goal 1: GMS enables accelerating the state of the art

CSCS Cray Piz Daint, 64 GB per compute node

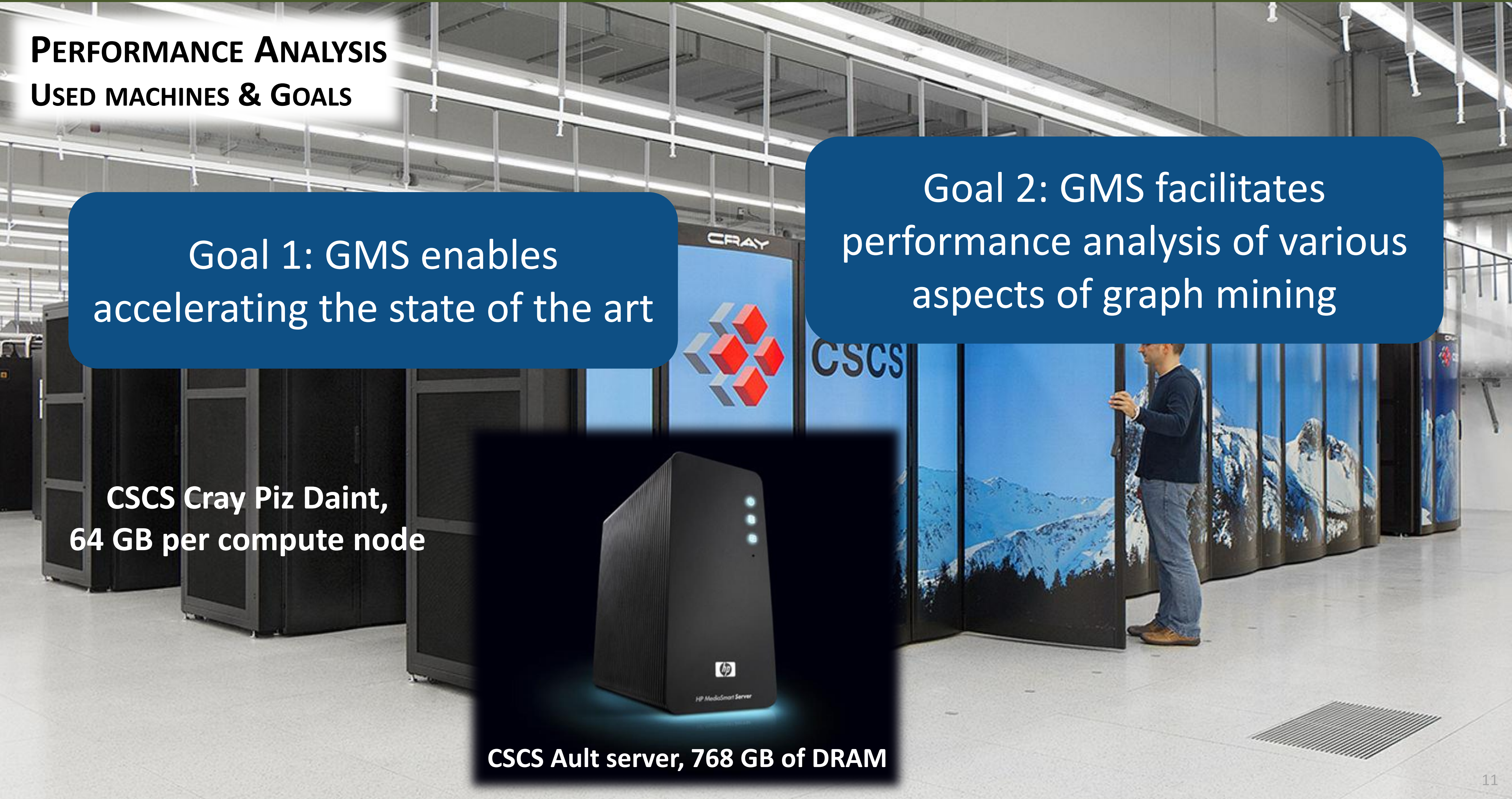
CSCS Ault server, 768 GB of DRAM

Performance Analysis

Used machines & Goals

Goal 1: GMS enables accelerating the state of the art

Goal 2: GMS facilitates performance analysis of various aspects of graph mining



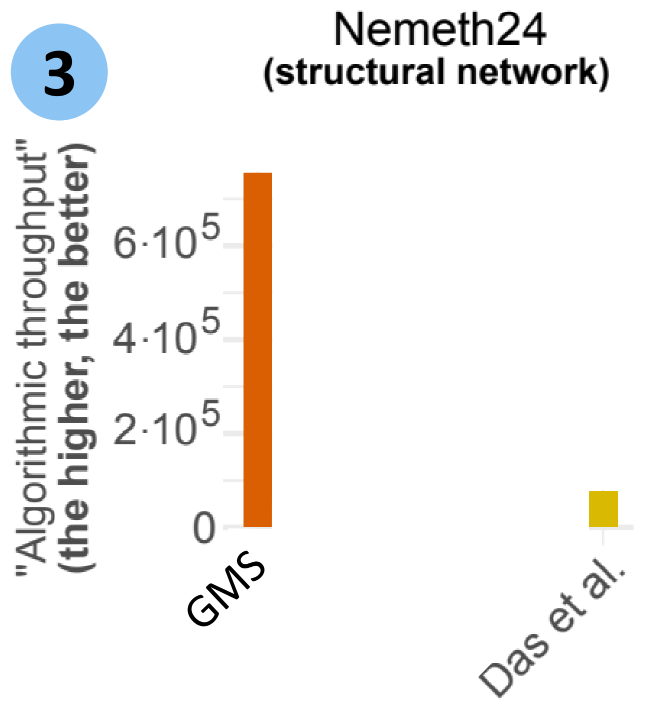
CSCS Cray Piz Daint, 64 GB per compute node

CSCS Ault server, 768 GB of DRAM

Goal 1: Accelerate the State-of-the-Art

Problem: Maximal Clique Listing **System**: Daint **Cores/threads**: 32

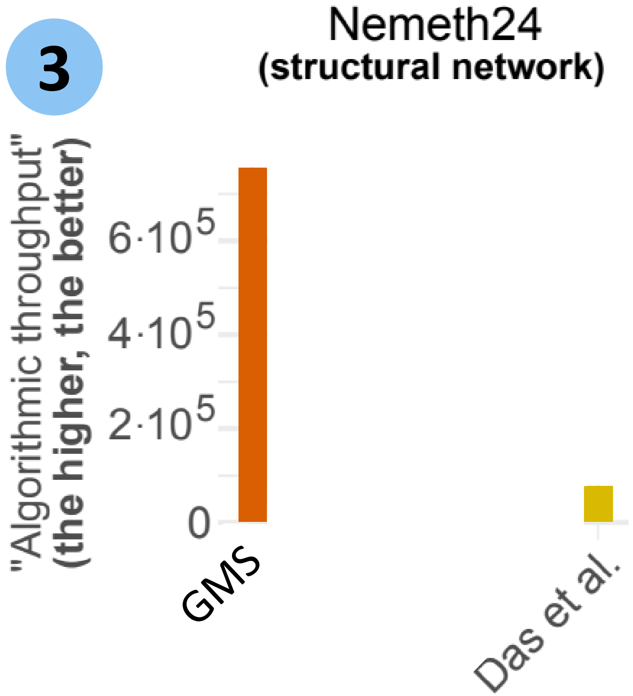
Goal 1: Accelerate the State-of-the-Art



Problem: Maximal Clique Listing **System**: Daint **Cores/threads**: 32

Goal 1: Accelerate the State-of-the-Art ✓

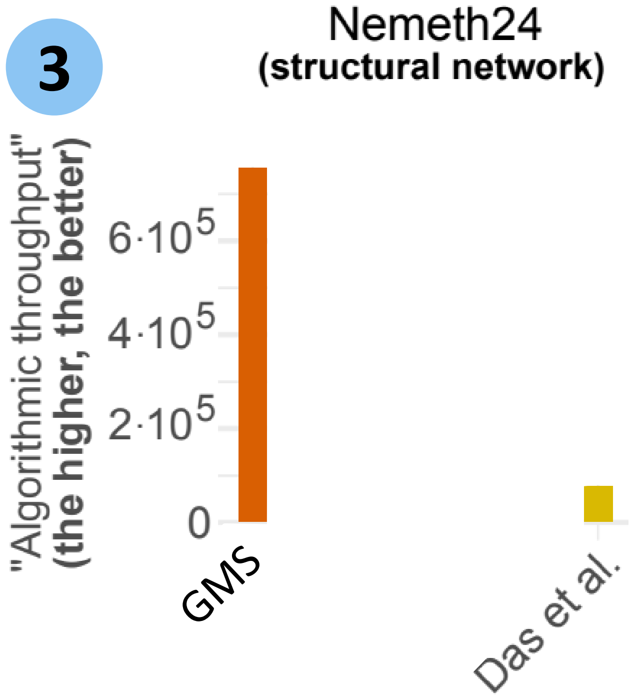
3



Problem: Maximal Clique Listing **System**: Daint **Cores/threads**: 32

Goal 1: Accelerate the State-of-the-Art ✓

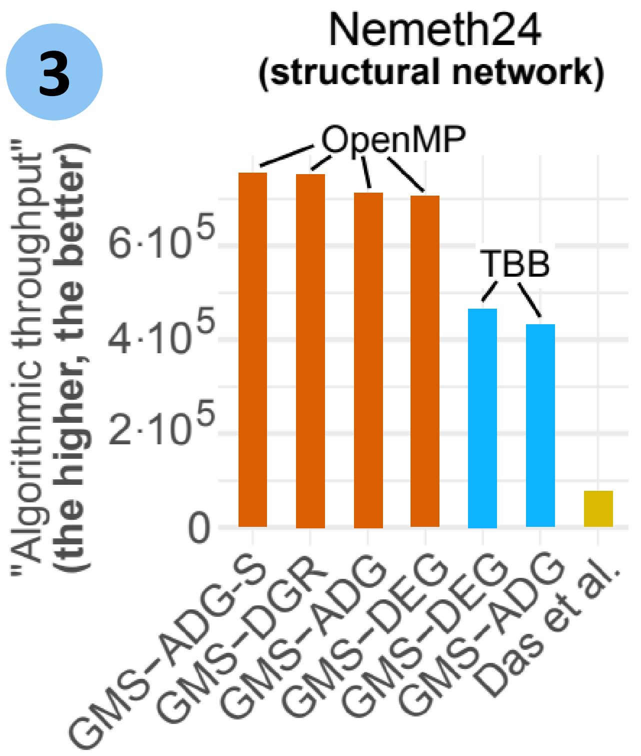
Goal 2: Facilitate Analysis



Problem: Maximal Clique Listing **System**: Daint **Cores/threads**: 32

Goal 1: Accelerate the State-of-the-Art ✓

Goal 2: Facilitate Analysis



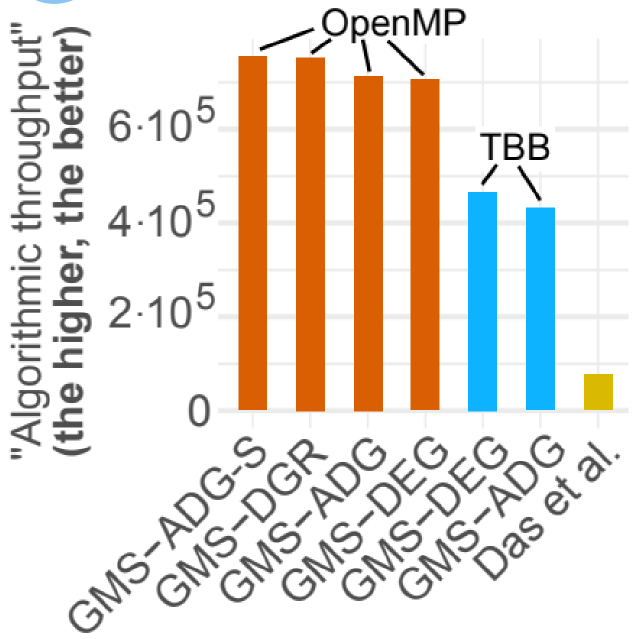
🟧 BK with the **GMS** code, OpenMP 🟦 BK with the **GMS** code, Intel TBB 🟨 BK by Das et al. (a recent baseline)

Problem: Maximal Clique Listing **System**: Daint **Cores/threads**: 32

Goal 1: Accelerate the State-of-the-Art ✓

Goal 2: Facilitate Analysis

3



Nemeth24
(structural network)

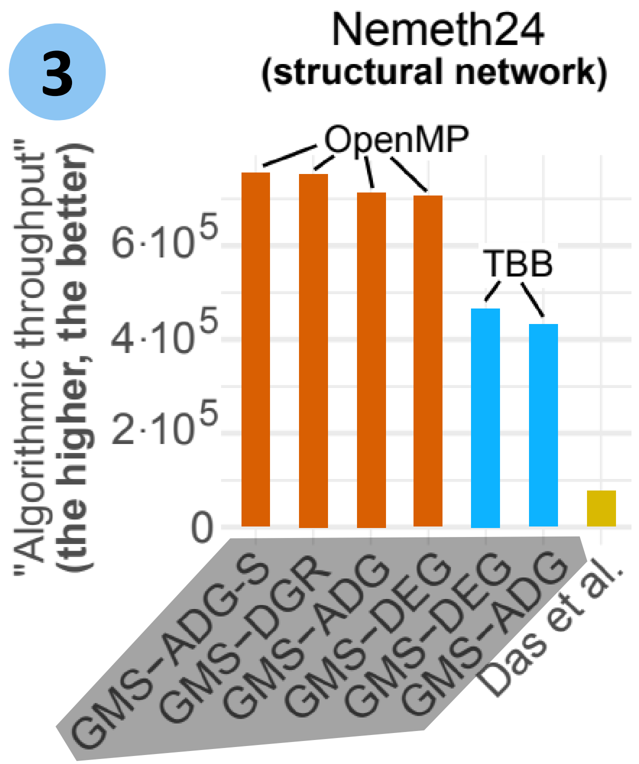
🟧 BK with the **GMS** code, OpenMP 🟦 BK with the **GMS** code, Intel TBB 🟨 BK by Das et al. (a recent baseline)

GMS-**DGR** : BK with degeneracy reordering (a variant by Eppstein et al.) GMS-**DEG** : BK with simple degree reordering
GMS-**ADG** : BK with approximate degeneracy reordering (a baseline obtained with GMS)
GMS-**ADG-S** : BK-GMS-ADG plus subgraph optimization (a baseline obtained with GMS)

Problem: Maximal Clique Listing **System**: Daint **Cores/threads**: 32

Goal 1: Accelerate the State-of-the-Art ✓

Goal 2: Facilitate Analysis



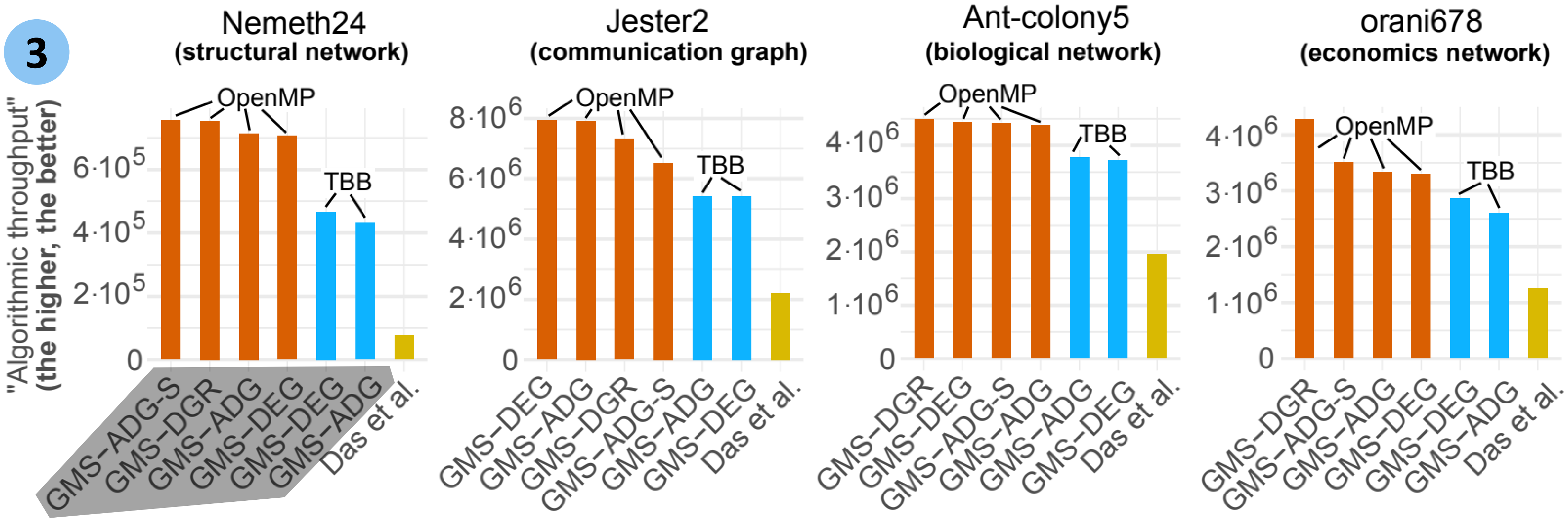
🟧 BK with the **GMS** code, OpenMP 🟦 BK with the **GMS** code, Intel TBB 🟨 BK by Das et al. (a recent baseline)

GMS-**DGR** : BK with degeneracy reordering (a variant by Eppstein et al.) GMS-**DEG** : BK with simple degree reordering
GMS-**ADG** : BK with approximate degeneracy reordering (a baseline obtained with GMS)
GMS-**ADG-S** : BK-GMS-ADG plus subgraph optimization (a baseline obtained with GMS)

Problem: Maximal Clique Listing **System**: Daint **Cores/threads**: 32

Goal 1: Accelerate the State-of-the-Art ✓

Goal 2: Facilitate Analysis

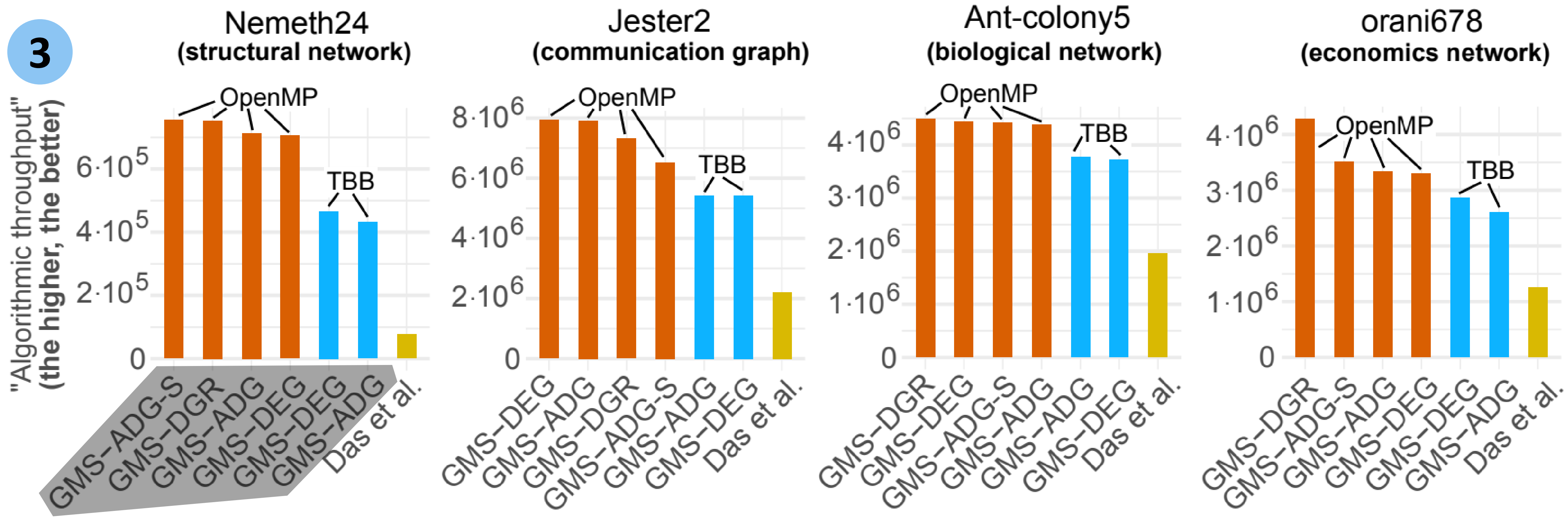


🟧 BK with the **GMS** code, OpenMP 🟦 BK with the **GMS** code, Intel TBB 🟨 BK by Das et al. (a recent baseline)

GMS-**DGR** : BK with degeneracy reordering (a variant by Eppstein et al.) GMS-**DEG** : BK with simple degree reordering
GMS-**ADG** : BK with approximate degeneracy reordering (a baseline obtained with GMS)
GMS-**ADG-S** : BK-GMS-ADG plus subgraph optimization (a baseline obtained with GMS)

Problem: Maximal Clique Listing **System**: Daint **Cores/threads**: 32

Goal 1: Accelerate the State-of-the-Art ✓ ✓ Goal 2: Facilitate Analysis


Nemeth24 (structural network)

Jester2 (communication graph)

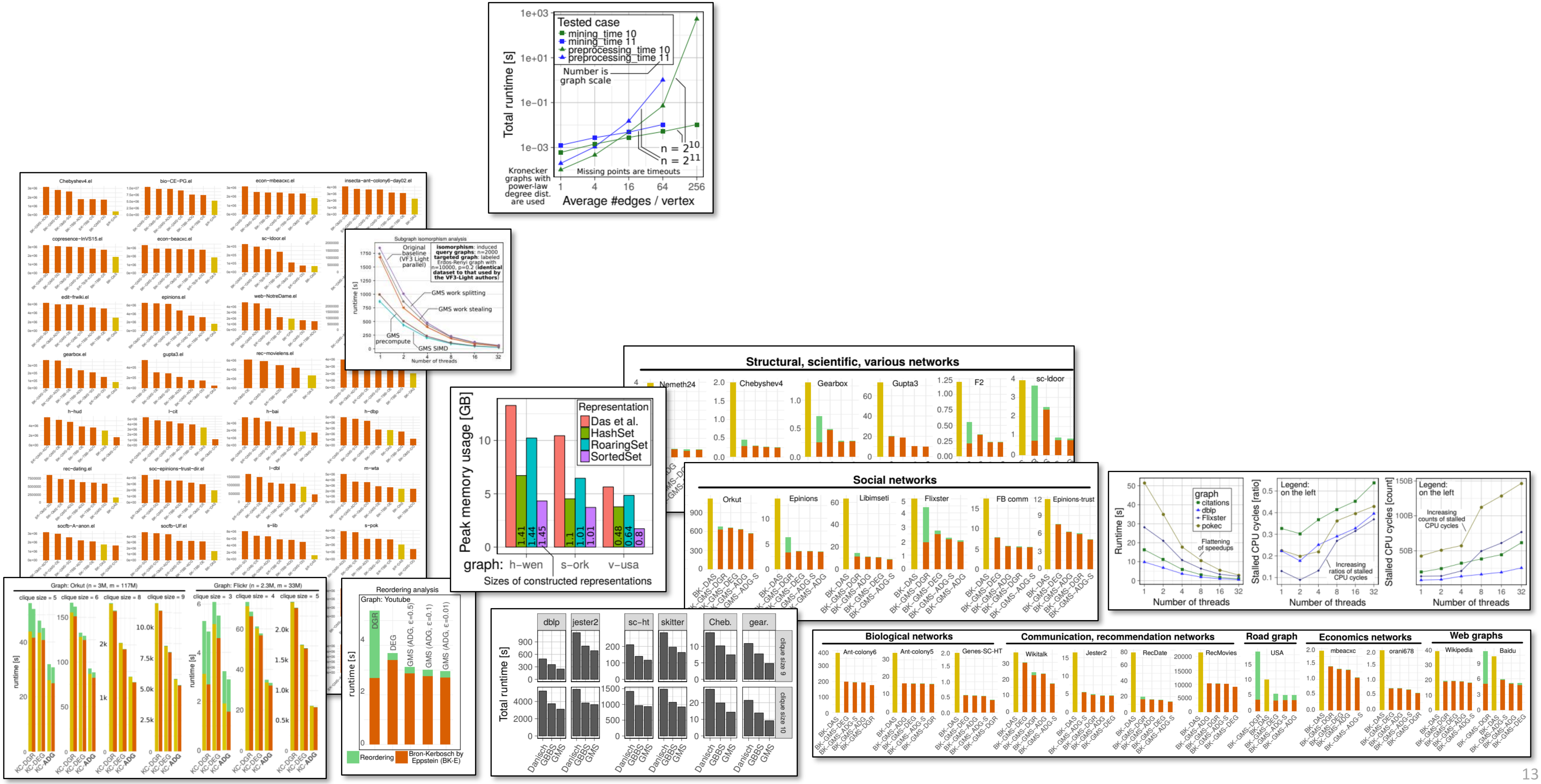
Ant-colony5 (biological network)

orani678 (economics network)

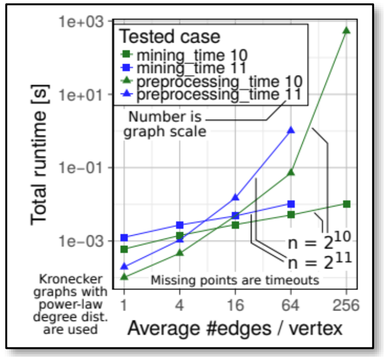
🟧 BK with the **GMS** code, OpenMP 🟦 BK with the **GMS** code, Intel TBB 🟨 BK by Das et al. (a recent baseline)

GMS-**DGR** : BK with degeneracy reordering (a variant by Eppstein et al.) GMS-**DEG** : BK with simple degree reordering
GMS-**ADG** : BK with approximate degeneracy reordering (a baseline obtained with GMS)
GMS-**ADG-S** : BK-GMS-ADG plus subgraph optimization (a baseline obtained with GMS)

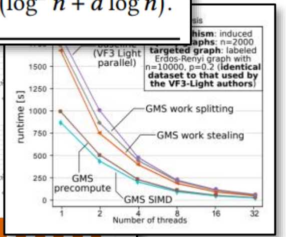
Problem: Maximal Clique Listing **System**: Daint **Cores/threads**: 32

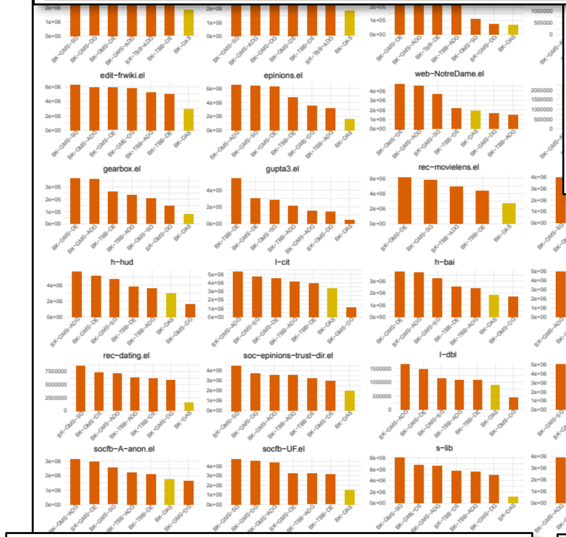

Tested case
mining_time 10
mining_time 11
preprocessing_time 10
preprocessing_time 11
Number is graph scale
n = 2^10
n = 2^11
Missing points are timeouts
Kronecker graphs with power-law degree dist. are used
Total runtime [s]
Average #edges / vertex
Subgraph isomorphism analysis
Original baseline (VF3 Light parallel)
isomorphism: induced query graphs: n=2000 targeted graph: labeled Erdos-Renyi graph with n=10000, p=0.2 (identical dataset to that used by the VF3-Light authors)
GMS work splitting
GMS work stealing
GMS precompute
GMS SIMD
Number of threads
Structural, scientific, various networks
Nemeth24
Chebyshev4
Gearbox
Gupta3
F2
sc-ldoor
Peak memory usage [GB]
Representation
Das et al.
HashSet
RoaringSet
SortedSet
graph: h-wen s-ork v-usa
Sizes of constructed representations
Social networks
Orkut
Epinions
Libimseti
Flixster
FB comm
Epinions-trust
Runtime [s]
Stalled CPU cycles [ratio]
Stalled CPU cycles [count]
graph
citations
dblp
Flixster
pokec
Flattening of speedups
Legend: on the left
Increasing ratios of stalled CPU cycles
Increasing counts of stalled CPU cycles
Number of threads
Graph: Orkut (n = 3M, m = 117M)
Graph: Flickr (n = 2.3M, m = 33M)
clique size = 5
clique size = 6
clique size = 8
clique size = 9
clique size = 3
clique size = 4
runtime [s]
Reordering analysis
Graph: Youtube
DGR
DEG
GMS (ADG, ε=0.5)
GMS (ADG, ε=0.1)
GMS (ADG, ε=0.01)
Reordering
Bron-Kerbosch by Eppstein (BK-E)
dblp
jester2
sc-ht
skitter
Cheb.
gear.
clique size 9
clique size 10
Danisch
GBBS
GMS
Biological networks
Communication, recommendation networks
Road graph
Economics networks
Web graphs
Ant-colony6
Ant-colony5
Genes-SC-HT
Wikitalk
Jester2
RecDate
RecMovies
USA
mbeacxc
orani678
Wikipedia
Baidu

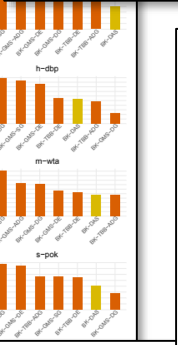
	Work	Depth
Chiba and Nishizeki [69]	$O\left(d^2 n(n-d)3^{d/3}\right)$	$O\left(d^2 n(n-d)3^{d/3}\right)$.
Chiba and Nishizeki [69]	$O\left(nd^{d+1}\right)$	$O\left(nd^{d+1}\right)$.
Chrobak and Eppstein [72]	$O\left(nd^2 2^d\right)$	$O\left(nd^2 2^d\right)$.
Eppstein et al. [92]	$O\left(dm\, 3^{\frac{d}{3}}\right)$	$O\left(dm\, 3^{\frac{d}{3}}\right)$.
Das et al. [79]	$O\left(3^{\frac{n}{3}}\right)$	$O\left(d \log n\right)$.
This Paper	$O\left(dm\, 3^{\frac{(2+\varepsilon)d}{3}}\right)$	$O\left(\log^2 n + d\log n\right)$.



Algorithm	AL (sorted)	AM	EL (unsorted)	EL (sorted)
Node Iterator (TC)	$O\left(n + m^{3/2}\log\Delta\right)^*$	$O\left(n + m^{3/2}\right)$	$O\left(n + m^{3/2}(\Delta + \log m)\right)$	$O\left(n + m^{5/2}\right)$
Rank Merge (TC)	$O\left(n + n\Delta + m^{3/2}\right)$	$O\left(n + n\Delta + m^{3/2}\right)$	$O\left(n + n\Delta + m^{3/2}\right)$	$O\left(n + n\Delta + m^{3/2}\right)$
BFS, top-down	$\Theta(n+m)$	$\Theta(n+m)$	$O(n\log m + m)$	$O(nm+n+m)$
PageRank, pushing	$O\left(n + m^{3/2}\log\Delta\right)^*$	$O\left(n + m^{3/2}\right)$	$O\left(n + m^{3/2}(\Delta + \log m)\right)$	$O\left(n + m^{5/2}\right)$
D–Stepping (SSSP)	$O\left(n + m + \frac{L}{D} + n_D + m_D\right)$	$O\left(n^2 + \frac{L}{D} + nn_D + m_D\right)$	$O\left(nm + \frac{L}{D} + n_D(\log m + \Delta) + m_D\right)$	$O\left(nm + m + \frac{L}{D} + n_D m + m_D\right)$
Bellman-Ford (SSSP)	$O(n^2+nm)$	$O(n^3)$	$O(n+nm)$	$O(n+nm)$
Boruvka (MST)	$O(m\log n)$	$O(n^2\log n)$	$O(nm\log n\log m)$	$O(n^2 m)$
Boman (Graph Coloring)	$O(n+m)$	$O(n^2)$	$O(n^2)$	$O(n+nm)$
Betweenness Centrality	$O(nm)$	$O(n^3)$	$O(nm\log m)$	$O(nm^2)$

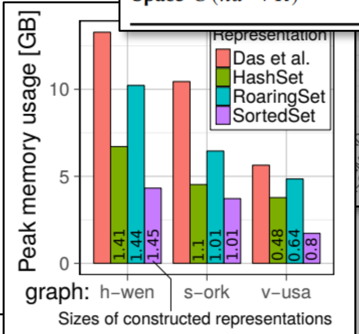


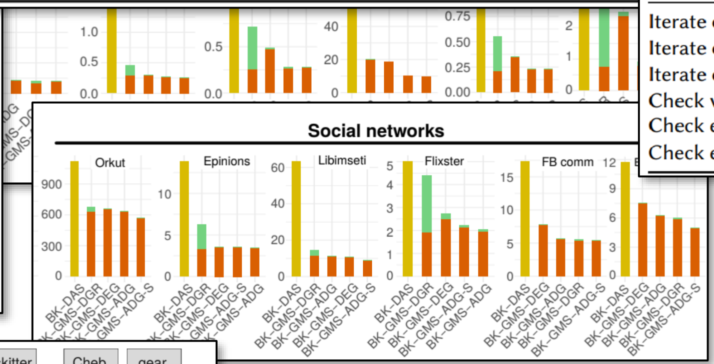


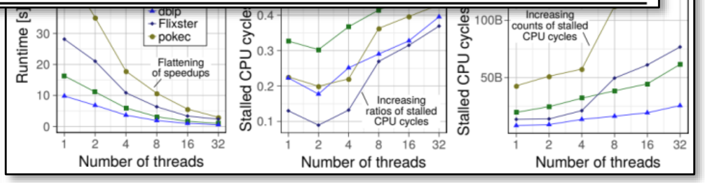


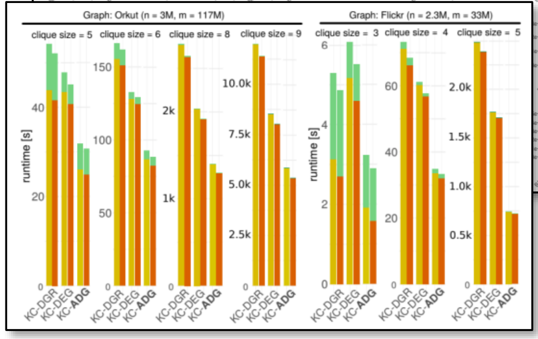
	k-Clique Listing *Node Parallel* [78]	k-Clique Listing *Edge Parallel* [78]	★ **k-Clique Listing with ADG (§ 6.3)**	ADG (Section 6)	Max. Cliques Eppstein et al. [91]	Max. Cliques Das et al. [79]	★ **Max. Cliques with ADG (§ 7.3)**	Subgr. Isomorphism *Node Parallel* [58, 75]	Link Prediction†, JP Clustering
Work	$O\left(mk\left(\frac{d}{2}\right)^{k-2}\right)$	$O\left(mk\left(\frac{d}{2}\right)^{k-2}\right)$	$O\left(mk\left(d+\frac{\varepsilon}{2}\right)^{k-2}\right)$	$O(m)$	$O\left(dm3^{d/3}\right)$	$O\left(3^{n/3}\right)$	$O\left(dm3^{(2+\varepsilon)d/3}\right)$	$O\left(n\Delta^{k-1}\right)$	$O(m\Delta)$
Depth	$O\left(n+k\left(\frac{d}{2}\right)^{k-1}\right)$	$O\left(n+k\left(\frac{d}{2}\right)^{k-2}+d^2\right)$	$O\left(k\left(d+\frac{\varepsilon}{2}\right)^{k-2}+\log^2 n+d^2\right)$	$O\left(\log^2 n\right)$	$O\left(dm3^{d/3}\right)$	$O(d\log n)$	$O\left(\log^2 n + d\log n\right)$	$O\left(\Delta^{k-1}\right)$	$O(\Delta)$
Space	$O(nd^2+K)$	$O\left(md^2+K\right)$	$O\left(md^2+K\right)$	$O(m)$	$O(m$				

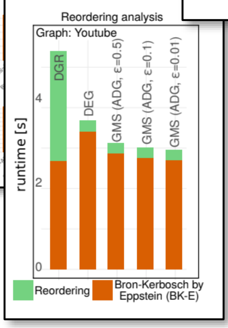
Graph query	**AL**	**AM**	**EL (unsorted)**	**EL (sorted)**
Iterate over all vertices	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
Iterate over all edges	$\Theta(n+m)$	$\Theta(n^2)$	$\Theta(m)$	$\Theta(m)$
Iterate over a neighborhood	$\Theta(\Delta)$	$\Theta(n)$	$\Theta(m)$	$\Theta(\log m + \Delta)^\#$
Check vertex' degree	$\Theta(n)^*$	$\Theta(n)^*$	$\Theta(m)^*$	$\Theta(\log m + \Delta)^{*\#}$
Check edge's existence	$O(\log\Delta)$	$O(1)$	$O(m)$	$O(\log m)$
Check edge's weight	$O(1)$	$O(n)$	$\Theta(m)$	$\Theta(\log m + \Delta)^\#$

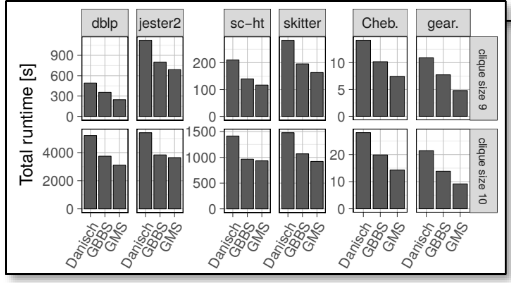


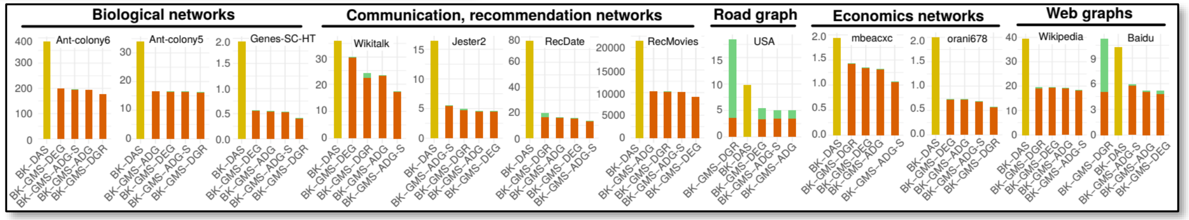


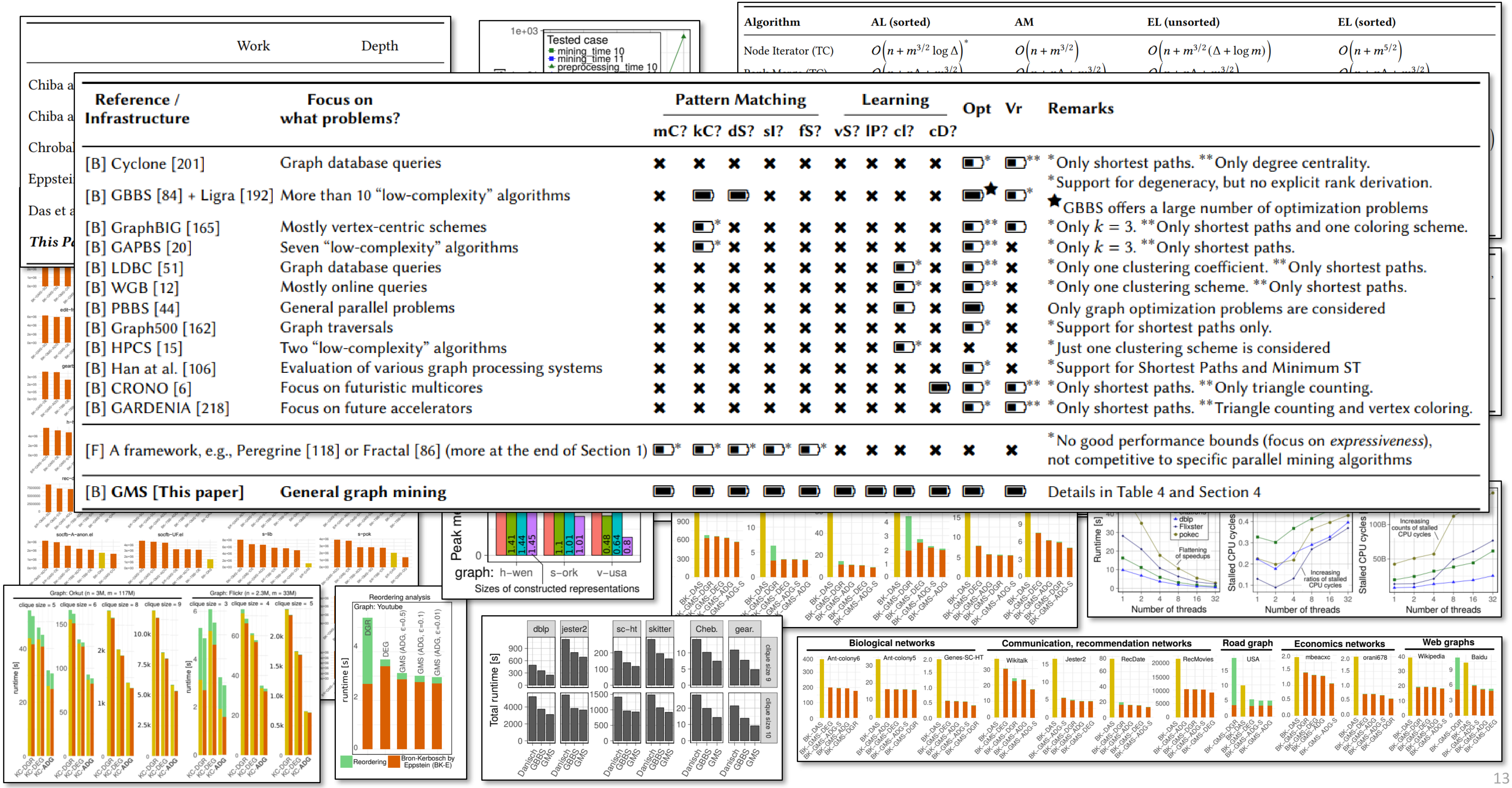






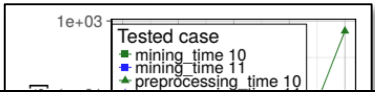






Reference / Infrastructure	Focus on what problems?	Pattern Matching mC?	kC?	dS?	sI?	fS?	Learning vS?	lP?	cl?	cD?	Opt	Vr	Remarks
[B] Cyclone [201]	Graph database queries	✖	✖	✖	✖	✖	✖	✖	✖	✖	◐*	◐**	*Only shortest paths. **Only degree centrality.
[B] GBBS [84] + Ligra [192]	More than 10 "low-complexity" algorithms	✖	●	●	✖	✖	✖	✖	✖	✖	●★	◐*	*Support for degeneracy, but no explicit rank derivation. ★GBBS offers a large number of optimization problems
[B] GraphBIG [165]	Mostly vertex-centric schemes	✖	◐*	✖	✖	✖	✖	✖	✖	✖	◐**	◐	*Only $k = 3$. **Only shortest paths and one coloring scheme.
[B] GAPBS [20]	Seven "low-complexity" algorithms	✖	◐*	✖	✖	✖	✖	✖	✖	✖	◐**	✖	*Only $k = 3$. **Only shortest paths.
[B] LDBC [51]	Graph database queries	✖	✖	✖	✖	✖	✖	✖	◐*	✖	◐**	✖	*Only one clustering coefficient. **Only shortest paths.
[B] WGB [12]	Mostly online queries	✖	✖	✖	✖	✖	✖	✖	◐*	✖	◐**	✖	*Only one clustering scheme. **Only shortest paths.
[B] PBBS [44]	General parallel problems	✖	✖	✖	✖	✖	✖	✖	◐	✖	●	✖	Only graph optimization problems are considered
[B] Graph500 [162]	Graph traversals	✖	✖	✖	✖	✖	✖	✖	✖	✖	◐*	✖	*Support for shortest paths only.
[B] HPCS [15]	Two "low-complexity" algorithms	✖	✖	✖	✖	✖	✖	✖	◐*	✖	✖	✖	*Just one clustering scheme is considered
[B] Han at al. [106]	Evaluation of various graph processing systems	✖	✖	✖	✖	✖	✖	✖	✖	✖	◐*	✖	*Support for Shortest Paths and Minimum ST
[B] CRONO [6]	Focus on futuristic multicores	✖	✖	✖	✖	✖	✖	✖	✖	●	◐*	◐**	*Only shortest paths. **Only triangle counting.
[B] GARDENIA [218]	Focus on future accelerators	✖	✖	✖	✖	✖	✖	✖	✖	✖	◐*	◐**	*Only shortest paths. **Triangle counting and vertex coloring.
[F] A framework, e.g., Peregrine [118] or Fractal [86] (more at the end of Section 1)		◐*	◐*	◐*	◐*	◐*	✖	✖	✖	✖	✖	✖	*No good performance bounds (focus on *expressiveness*), not competitive to specific parallel mining algorithms
[B] GMS [This paper]	**General graph mining**	●	●	●	●	●	●	●	●	●	●	●	Details in Table 4 and Section 4

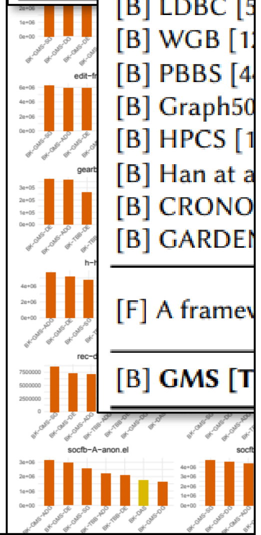
	Work	Depth
Chiba a		
Chiba a		
Chrobal		
Epsteii		
Das et a		
This Pa		

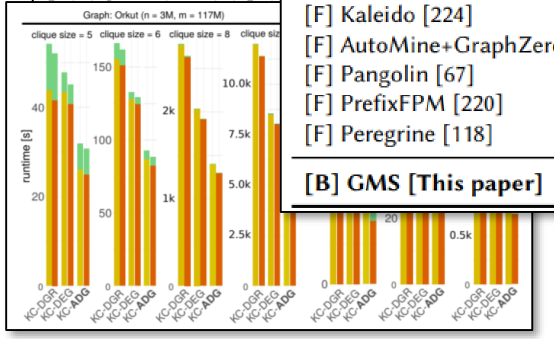


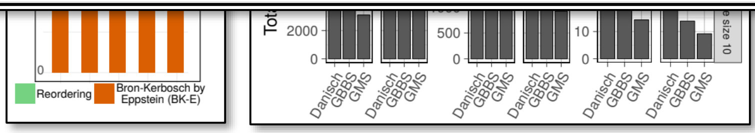
Algorithm	AL (sorted)	AM	EL (unsorted)	EL (sorted)
Node Iterator (TC)	$O\left(n + m^{3/2} \log \Delta\right)^*$	$O\left(n + m^{3/2}\right)$	$O\left(n + m^{3/2} (\Delta + \log m)\right)$	$O\left(n + m^{5/2}\right)$

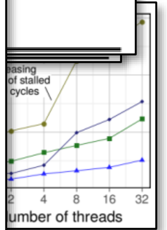
Reference / Infrastructure	Focus on what problems?	Pattern Matching					Learning				Opt	Vr	Remarks
		mC?	kC?	dS?	sI?	fS?	vS?	lP?	cI?	cD?			
[B] Cyclone													on.
[B] GBBS [8													ms
[B] GraphBI													scheme.
[B] GAPBS													ns.
[B] LDBC [5													
[B] WGB [1													
[B] PBBS [4													
[B] Graph50													
[B] HPCS [1													
[B] Han at a													
[B] CRONO													coloring.
[B] GARDEN													
[F] A framew													*ss*),
[B] GMS [T													

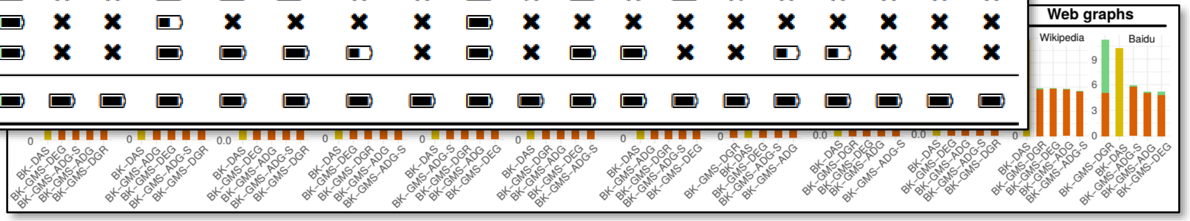
Reference / Infrastructure	Summary of focus (functionalities)	New Alg			Gen. APIs				Metrics					Storage				Compres.					Th.	
		∃	na	sp	N	G	S	P	rt	me	fg	mf	af	ag	bg	aa	ba	ad	of	fg	en	re	∃	nb
[B] Cyclone [201]	Graph databases	✗	✗	✗	✗	◐	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[B] GBBS [84] + Ligra [192]	General graph processing	✗	✗	✗	◐	●	●	✗	●	◐	✗	✗	✗	●	●	●	●	●	◐	✗	◐	✗	●	●
[B] GraphBIG [165]	General graph processing	✗	✗	✗	●	◐	✗	✗	●	●	✗	●	✗	●	✗	●	✗	✗	✗	✗	✗	✗	✗	✗
[B] GAPBS [20]	General graph processing	✗	✗	✗	✗	◐	✗	✗	●	◐	✗	✗	✗	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[B] Graphalytics LDBC [51]	Graph databases	✗	✗	✗	✗	◐	✗	✗	●*	●*	◐*	●*	◐*	●	✗	●	✗	✗	✗	✗	✗	✗	✗	✗
[B] WGB [12]	General graph processing	✗	✗	✗	✗	◐	✗	✗	◐	◐	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[B] PBBS [44]	General graph processing	✗	✗	✗	✗	✗	✗	✗	◐	✗	✗	✗	✗	◐	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[B] Graph500 [162]	Graph traversals	●	◐	●	✗	◐	✗	✗	●	✗	✗	✗	◐	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[B] HPCS [15]	General graph processing	✗	✗	✗	✗	◐	✗	✗	◐	◐	✗	✗	✗	●	✗	✗	✗	✗	✗	✗	✗	✗	◐	◐
[B] Han et al. [106]	Evaluation of graph processing systems	✗	✗	✗	◐	✗	✗	✗	●	●	✗	◐	✗	◐	◐	✗	✗	✗	✗	✗	✗	✗	✗	✗
[B] CRONO [6]	Multicore systems	✗	✗	✗	✗	✗	✗	✗	●	●	◐	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[B] GARDENIA [218]	Accelerators	✗	✗	✗	✗	◐	✗	✗	●	◐	✗	●	✗	◐	✗	◐	✗	✗	✗	✗	✗	✗	✗	✗
[F] Arabesque [204]	Graph pattern matching	●	◐	◐	✗	●	✗	✗	●	●	●	✗	✗	●	✗	●	✗	●	✗	✗	✗	✗	●	✗
[F] NScale [174]	Ego-network analysis	●	◐	●	✗	●	✗	✗	●	●	●	✗	✗	●	✗	✗	●	●	✗	✗	✗	✗	●	✗
[F] G-Thinker [219]	Graph pattern matching	●	✗	●	✗	●	✗	◐	●	◐	◐	✗	✗	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[F] G-Miner [66]	Graph pattern matching	●	◐	●	✗	◐	✗	✗	●	●	◐*	◐*	✗	●	✗	●	✗	✗	✗	✗	✗	✗	●	✗
[F] Nuri [124]	Graph pattern matching	●	✗	●	✗	●	✗	✗	●	◐*	◐*	✗	✗	●	✗	●	✗	✗	✗	✗	✗	✗	✗	✗
[F] RStream [210]	Graph pattern matching	●	✗	◐	✗	◐	✗	✗	●	◐*	◐*	✗	✗	●	✗	●	✗	✗	✗	✗	✗	✗	✗	✗
[F] ASAP [116]	Graph pattern matching	●	◐	◐	✗	◐	✗	✗	●	◐*	◐*	✗	✗	◐	✗	✗	✗	✗	✗	✗	✗	✗	●	✗
[F] Fractal [86]	Graph pattern matching	●	✗	●	✗	●	✗	✗	●	●	●*	✗	✗	●	✗	●	✗	✗	✗	✗	✗	✗	✗	✗
[F] Kaleido [224]	Graph pattern matching	●	◐	◐	✗	◐	✗	✗	●	●	◐*	◐*	✗	●	✗	✗	●	●	✗	✗	✗	✗	●	✗
[F] AutoMine+GraphZero [153, 154]	Graph pattern matching	●	◐	◐	✗	◐	✗	●	●	●	✗	◐	✗	●	✗	✗	✗	✗	✗	✗	✗	✗	●	✗
[F] Pangolin [67]	Graph pattern matching	●	✗	◐	✗	●	✗	✗	●	●	◐*	✗	✗	●	✗	●	✗	●	✗	✗	✗	✗	✗	✗
[F] PrefixFPM [220]	Graph Pattern Mining	◐	✗	◐	✗	●	✗	✗	◐	✗	✗	✗	✗	●	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
[F] Peregrine [118]	Graph Pattern Mining	●	✗	●	✗	●	✗	✗	●	●	●	◐	✗	●	✗	●	●	✗	✗	◐	◐	✗	✗	✗
[B] GMS [This paper]	Graph mining algorithms	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●





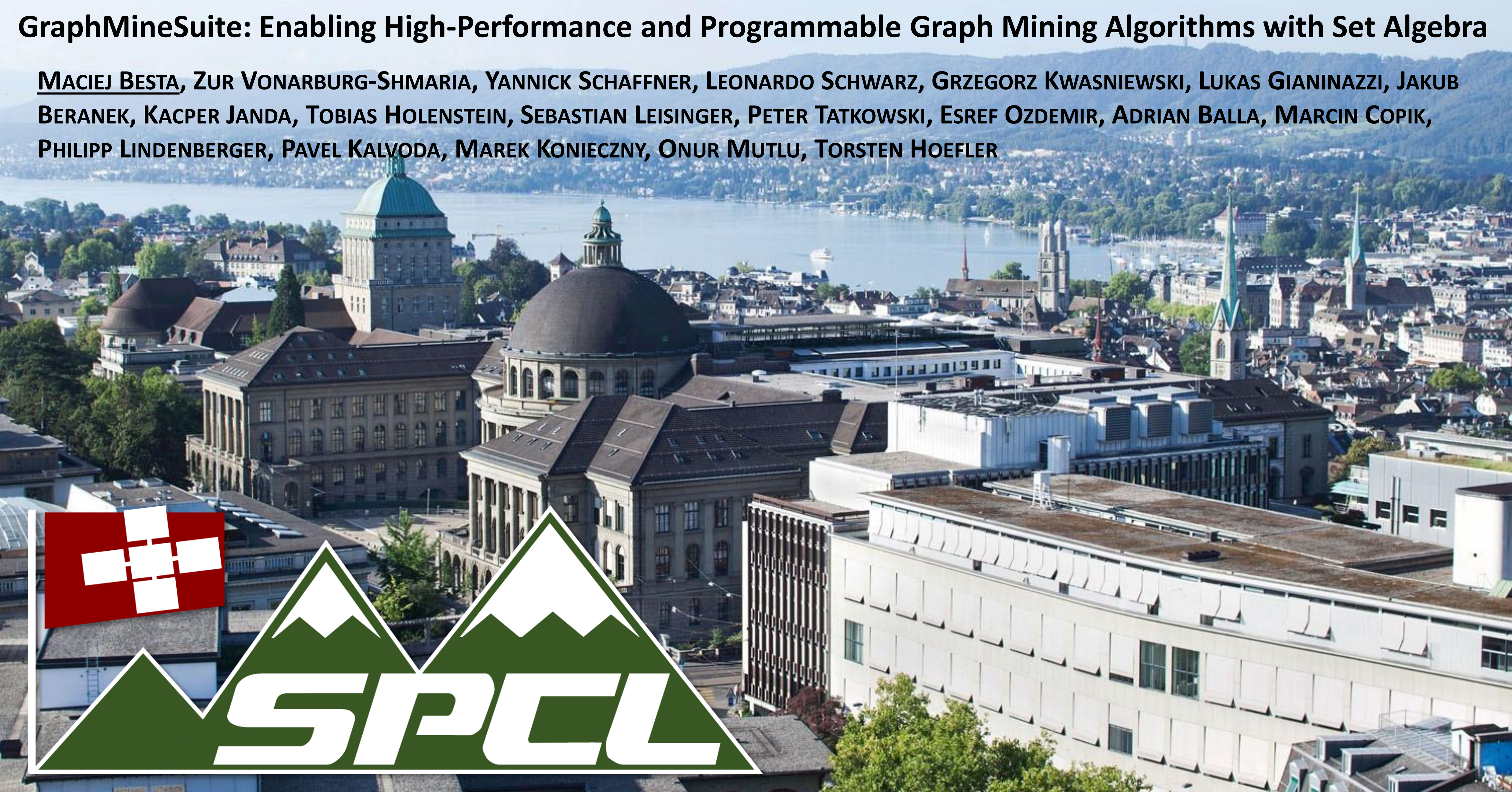






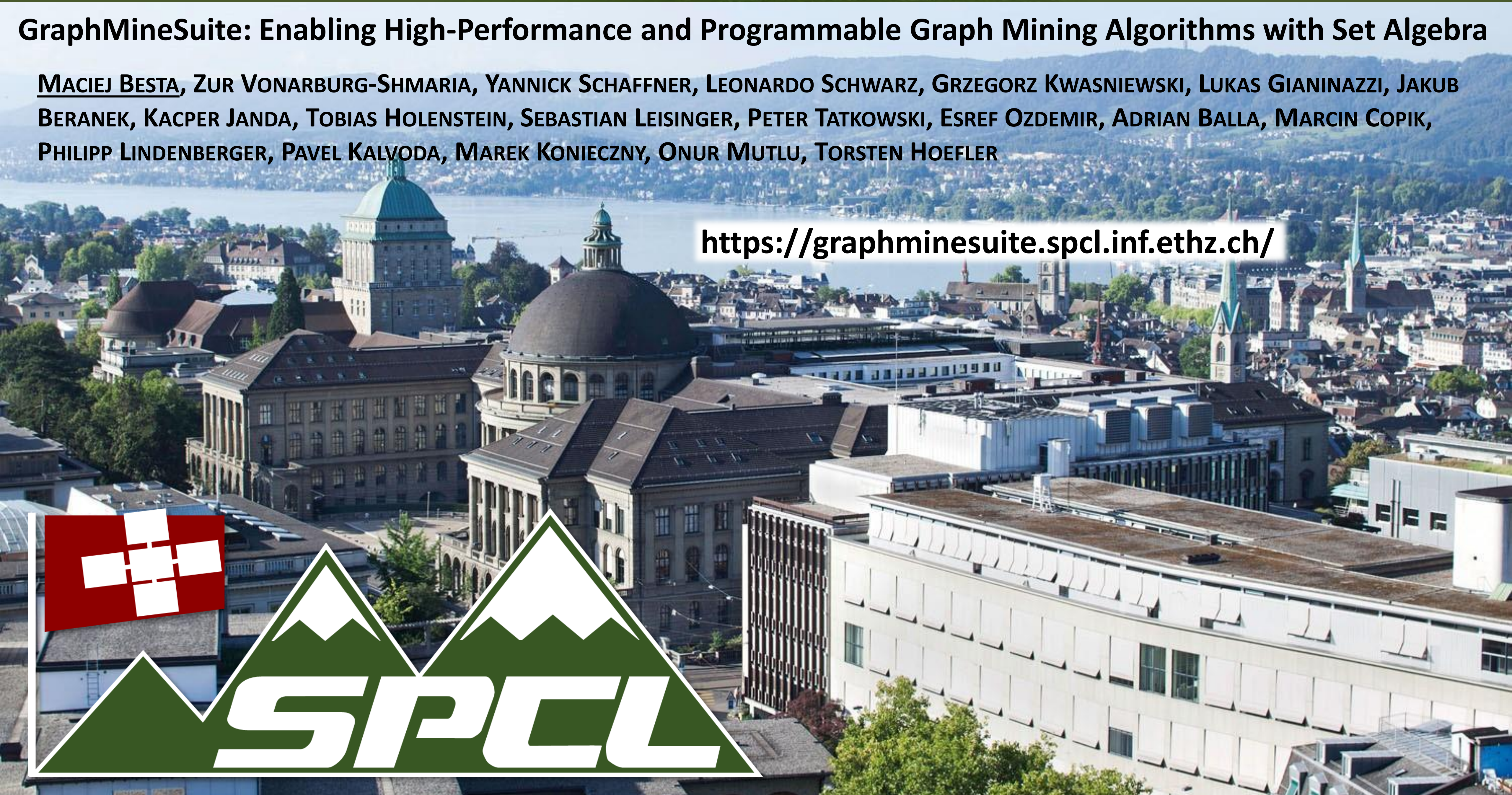
GraphMineSuite: Enabling High-Performance and Programmable Graph Mining Algorithms with Set Algebra

Maciej Besta, Zur Vonarburg-Shmaria, Yannick Schaffner, Leonardo Schwarz, Grzegorz Kwasniewski, Lukas Gianinazzi, Jakub Beranek, Kacper Janda, Tobias Holenstein, Sebastian Leisinger, Peter Tatkowski, Esref Ozdemir, Adrian Balla, Marcin Copik, Philipp Lindenberger, Pavel Kalvoda, Marek Konieczny, Onur Mutlu, Torsten Hoefler



GraphMineSuite: Enabling High-Performance and Programmable Graph Mining Algorithms with Set Algebra

Maciej Besta, Zur Vonarburg-Shmaria, Yannick Schaffner, Leonardo Schwarz, Grzegorz Kwasniewski, Lukas Gianinazzi, Jakub Beranek, Kacper Janda, Tobias Holenstein, Sebastian Leisinger, Peter Tatkowski, Esref Ozdemir, Adrian Balla, Marcin Copik, Philipp Lindenberger, Pavel Kalvoda, Marek Konieczny, Onur Mutlu, Torsten Hoefler

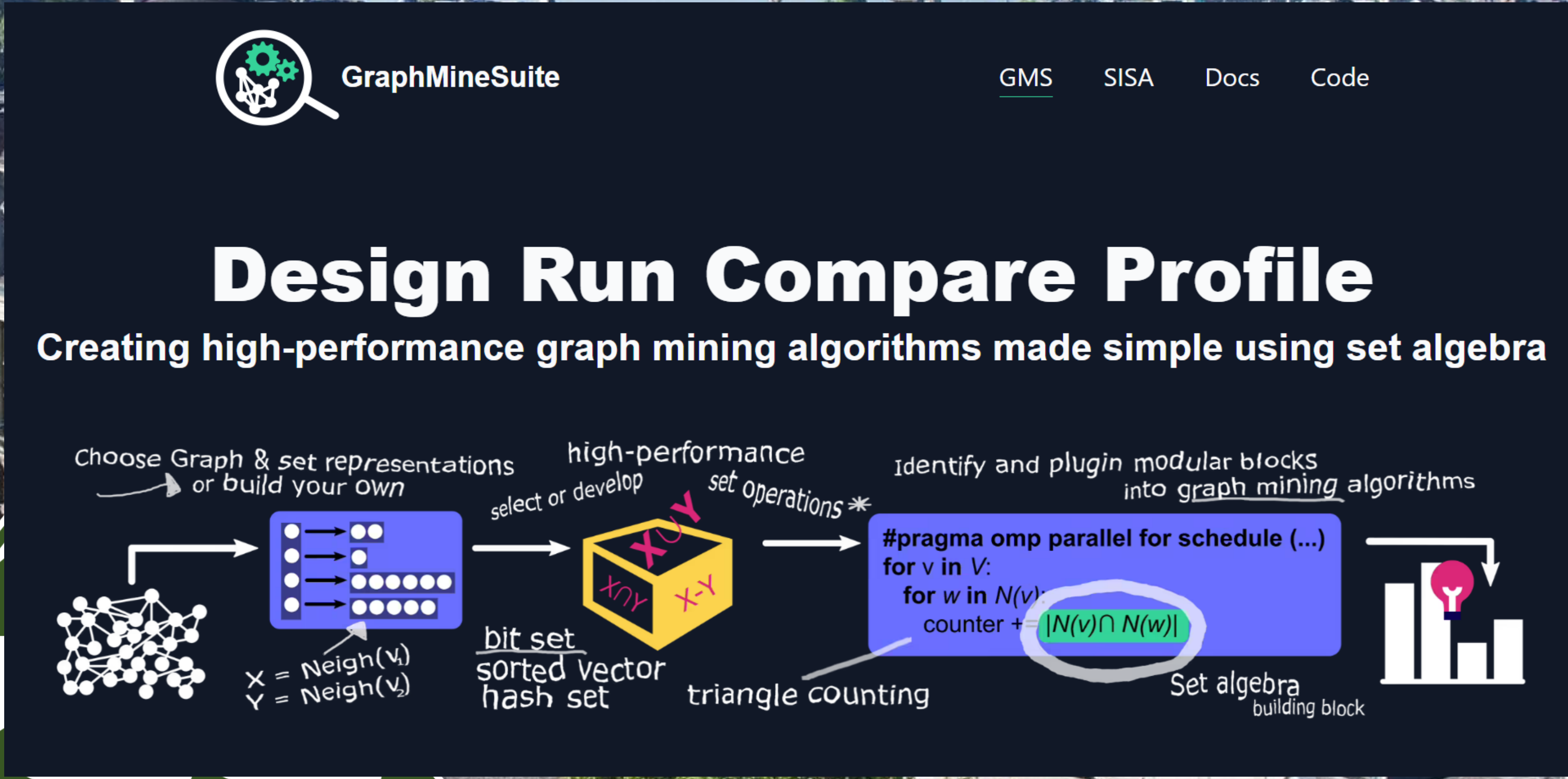


https://graphminesuite.spcl.inf.ethz.ch/

GraphMineSuite: Enabling High-Performance and Programmable Graph Mining Algorithms with Set Algebra

Maciej Besta, Zur Vonarburg-Shmaria, Yannick Schaffner, Leonardo Schwarz, Grzegorz Kwasniewski, Lukas Gianinazzi, Jakub Beranek, Kacper Janda, Tobias Holenstein, Sebastian Leisinger, Peter Tatkowski, Esref Ozdemir, Adrian Balla, Marcin Copik, Philipp Lindenberger, Pavel Kalvoda, Marek Konieczny, Onur Mutlu, Torsten Hoefler

https://graphminesuite.spcl.inf.ethz.ch/



GraphMineSuite

GMS SISA Docs Code

Design Run Compare Profile

Creating high-performance graph mining algorithms made simple using set algebra

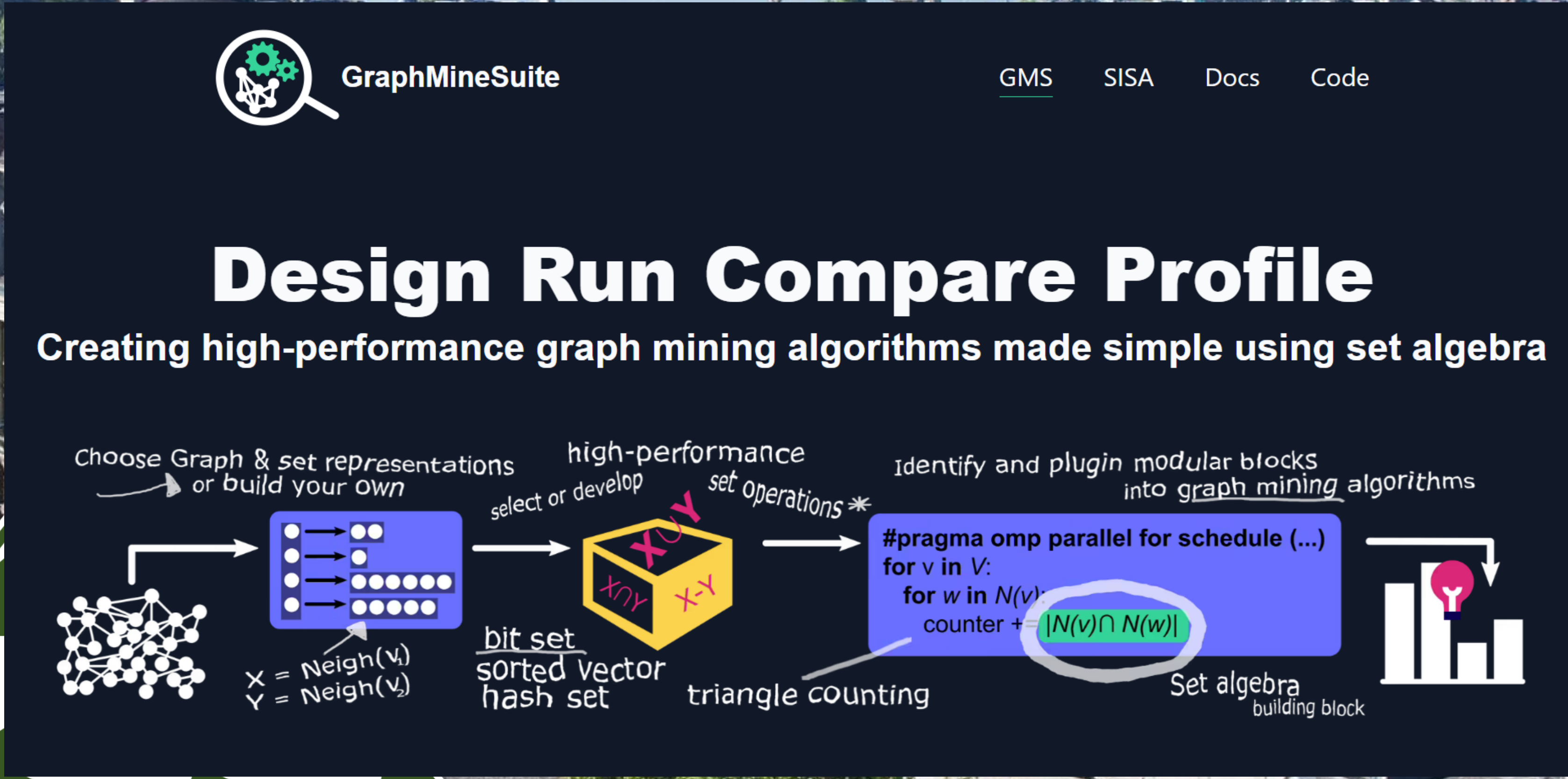
```
#pragma omp parallel for schedule (...)
for v in V:
  for w in N(v)
    counter += |N(v)∩ N(w)|
```

GraphMineSuite: Enabling High-Performance and Programmable Graph Mining Algorithms with Set Algebra

MACIEJ BESTA, ZUR VONARBURG-SHMARIA, YANNICK SCHAFFNER, LEONARDO SCHWARZ, GRZEGORZ KWASNIEWSKI, LUKAS GIANINAZZI, JAKUB BERANEK, KACPER JANDA, TOBIAS HOLENSTEIN, SEBASTIAN LEISINGER, PETER TATKOWSKI, ESREF OZDEMIR, ADRIAN BALLA, MARCIN COPIK, PHILIPP LINDENBERGER, PAVEL KALVODA, MAREK KONIECZNY, ONUR MUTLU, TORSTEN HOEFLER

https://graphminesuite.spcl.inf.ethz.ch/

Thank you for your attention



GraphMineSuite
GMS
SISA
Docs
Code

Design Run Compare Profile

Creating high-performance graph mining algorithms made simple using set algebra

Choose Graph & set representations or build your own
X = Neigh(v_1)
Y = Neigh(v_2)
select or develop high-performance set operations *
X∪Y
X∩Y
X-Y
bit set
sorted vector
hash set
Identify and plugin modular blocks into graph mining algorithms
triangle counting
Set algebra building block

```
#pragma omp parallel for schedule (...)
for v in V:
  for w in N(v):
    counter += |N(v)∩ N(w)|
```