Sublinear-Space Bounded-Delay Enumeration for Massive Network Analytics: Maximal Cliques

Alessio Conte, **Roberto Grossi**

Andrea Marino, *Luca Versari*

conte@di.unipi.it - luca.versari@sns.it

University of Pisa
Scuola Normale Superiore

Graph enumeration: does and donts

Given a graph list all the wanted solutions

- **Brute force**: test **all subsets** of vertices and check if they form a solution
- **Better way**: generate **only the wanted solutions**, possibly a in “smart” way

Why cliques?

Fundamental in **Network Analysis**

- It generalizes **triangles** (that show how clustered or "**social**" a network is)
- Used to define communities

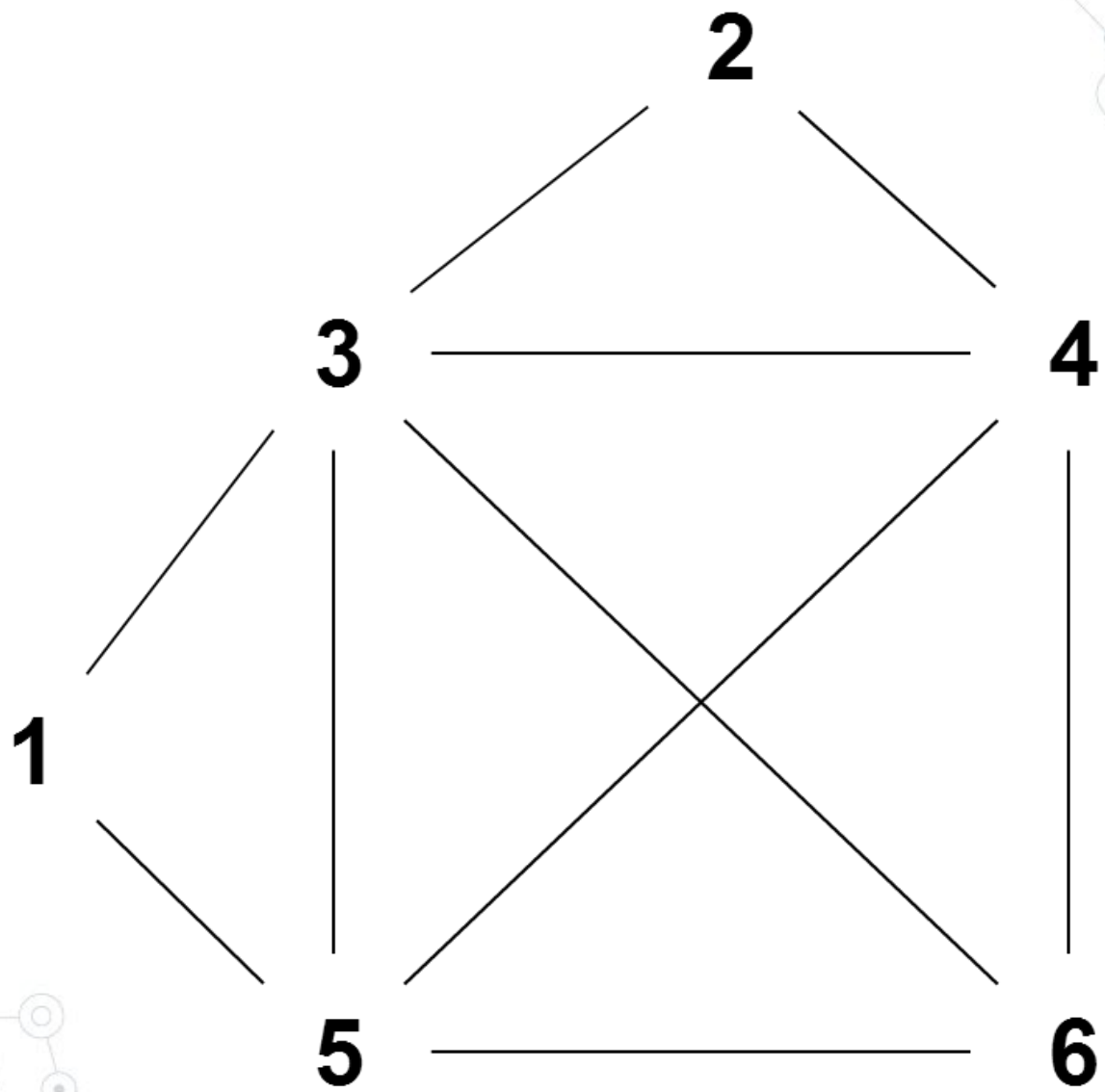
Notation

G: an undirected graph with **n** nodes and **m** edges.

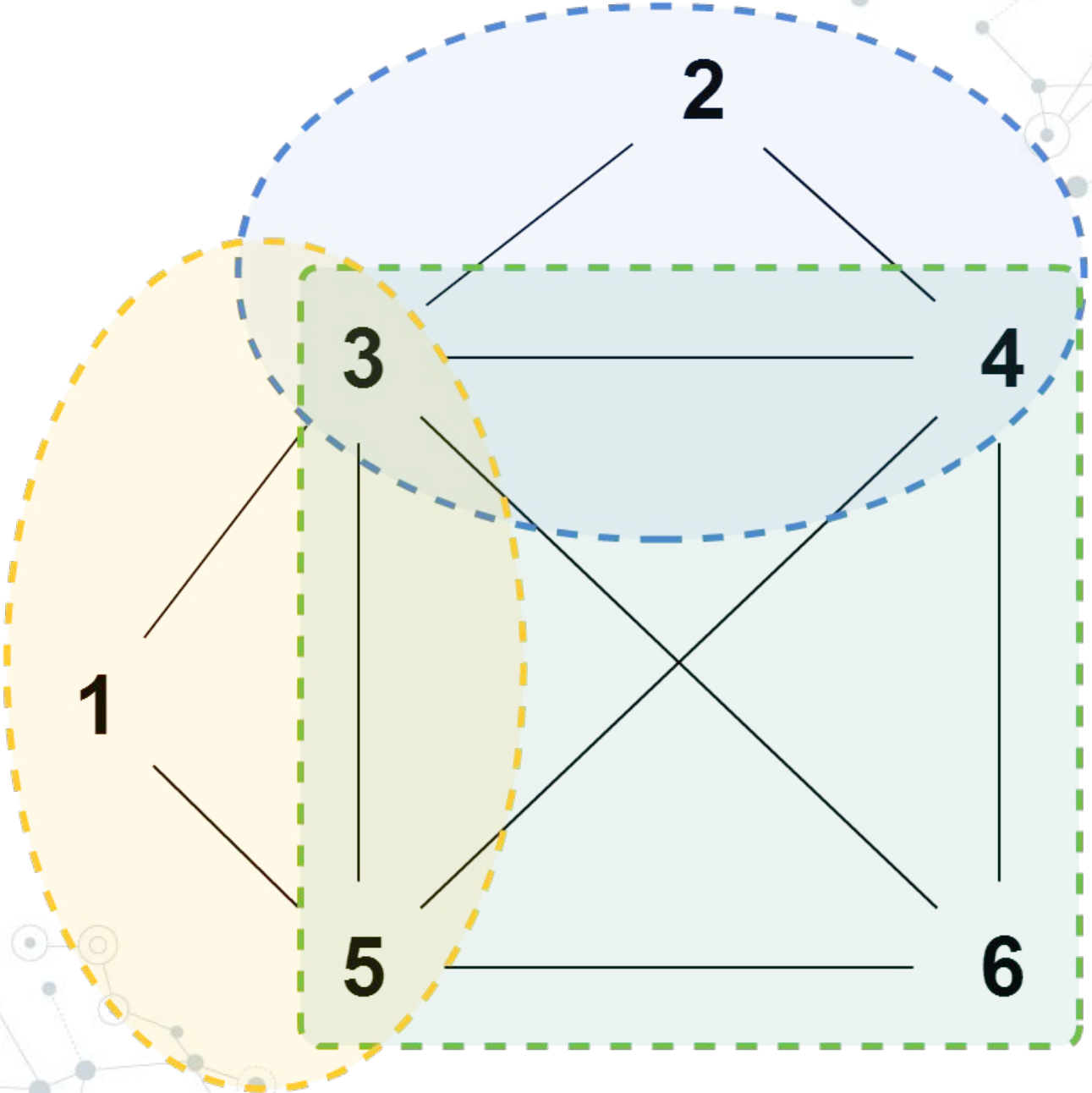
Clique: a set of nodes of **G** all pairwise connected

Maximal: if *not* properly contained in another clique

Input:



Output:



“Smart” enumeration:

Bounded-Delay:

Maximum time between two outputs is bounded by **poly(m,n)**

Sublinear-Space:

Use **o(|E|)** (small o) memory in addition to the input graph

Bounded-Delay:

Maximum time between two outputs is bounded by **poly(m,n)**

Sublinear-Space:

Use **o(|E|)** (small o) memory in addition to the input graph

Bounded-delay/ output-sensitive: why?

- ***Bounded-delay*** gives us ***output-sensitive*** algorithms (cost $\propto$ output)

- ***Output-sensitive*** algorithms are great when the number of solutions is variable

Some output-sensitive results:

Listing Maximal Acyclic Subgraphs:

[**Schwikowski *et al***] On enumerating all minimal solutions of feedback problems *DAM 2002*

Listing Trees:

[**Ruskey *et al***] Listing and Counting Subtrees of a Tree *SIAM J COMPUT 1981*

[**Shioura *et al***] An Optimal Algorithm for Scanning All Spanning Trees of undirected Graphs *SIAM J COMPUT 1997*

Listing Pseudo Cliques:

[**Uno**] An efficient algorithm for solving pseudo clique enumeration problem *ALGORITHMICA 2010*

Listing Paths/Cycles:

[**Eppstein *et al***] Finding the K Shortest Paths *SIAM J COMPUT 1998*

[**Birmelé *et al***] Optimal Listing of Cycles and st-Paths in Undirected Graphs *SODA 2012*

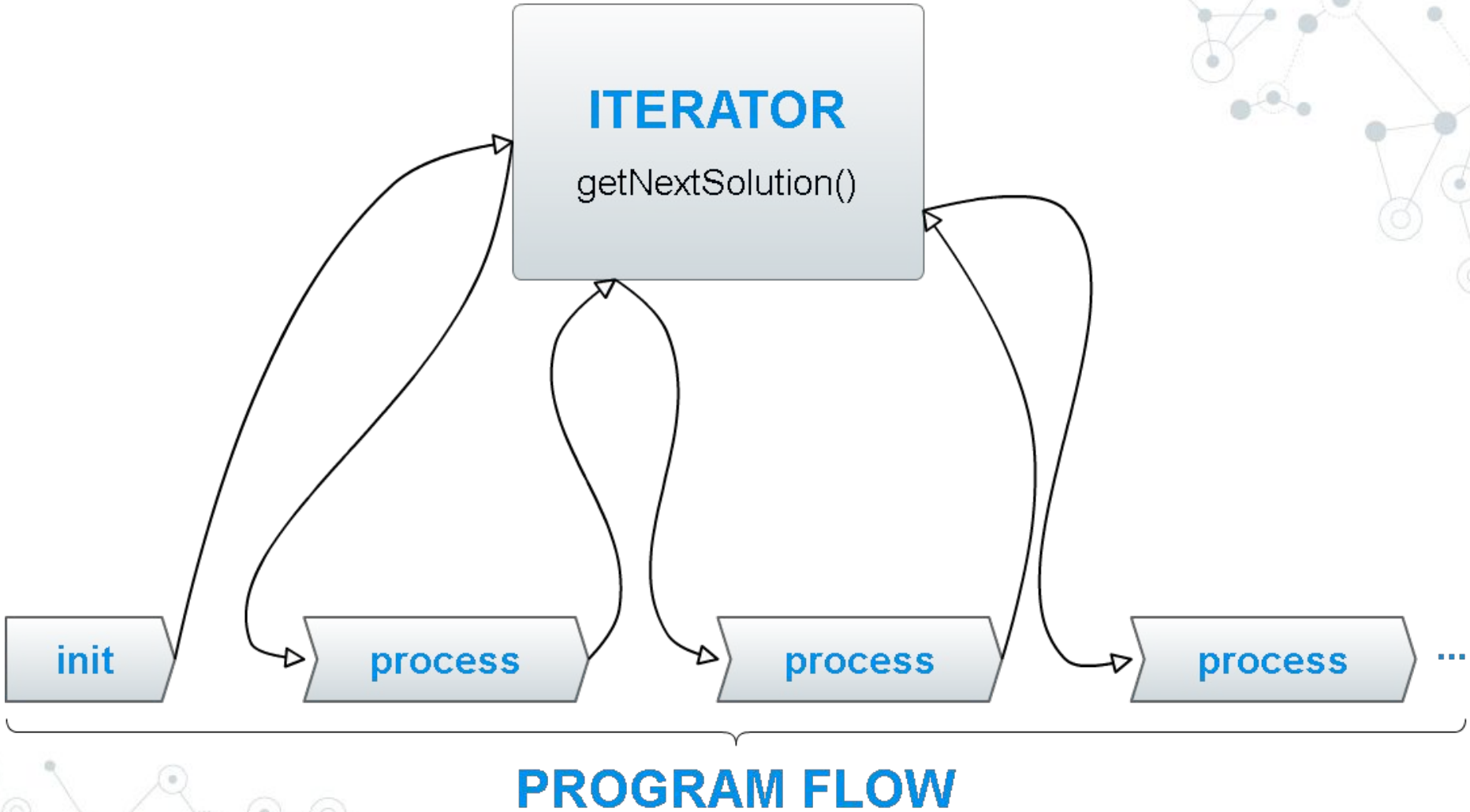
[**Johnson**] Finding All the Elementary Circuits of a Directed Graph *SIAM J COMPUT 1975*

Listing Triangles:

[**Bjørklund et al**] Listing Triangles *ICALP'14*

[**Kopelowitz et al**] Higher Lower Bounds from the 3SUM Conjecture *SODA'16*

Delay: a stronger constraint



Delay: a stronger constraint

The complexity of an iterator step is the ***delay*** of the algorithm

Bounded-Delay:

Maximum time between two outputs is bounded by **poly(m,n)**

Sublinear-Space:

Use **o(|E|)** (small o) memory in addition to the input graph

Advantages in modern multicore environments:

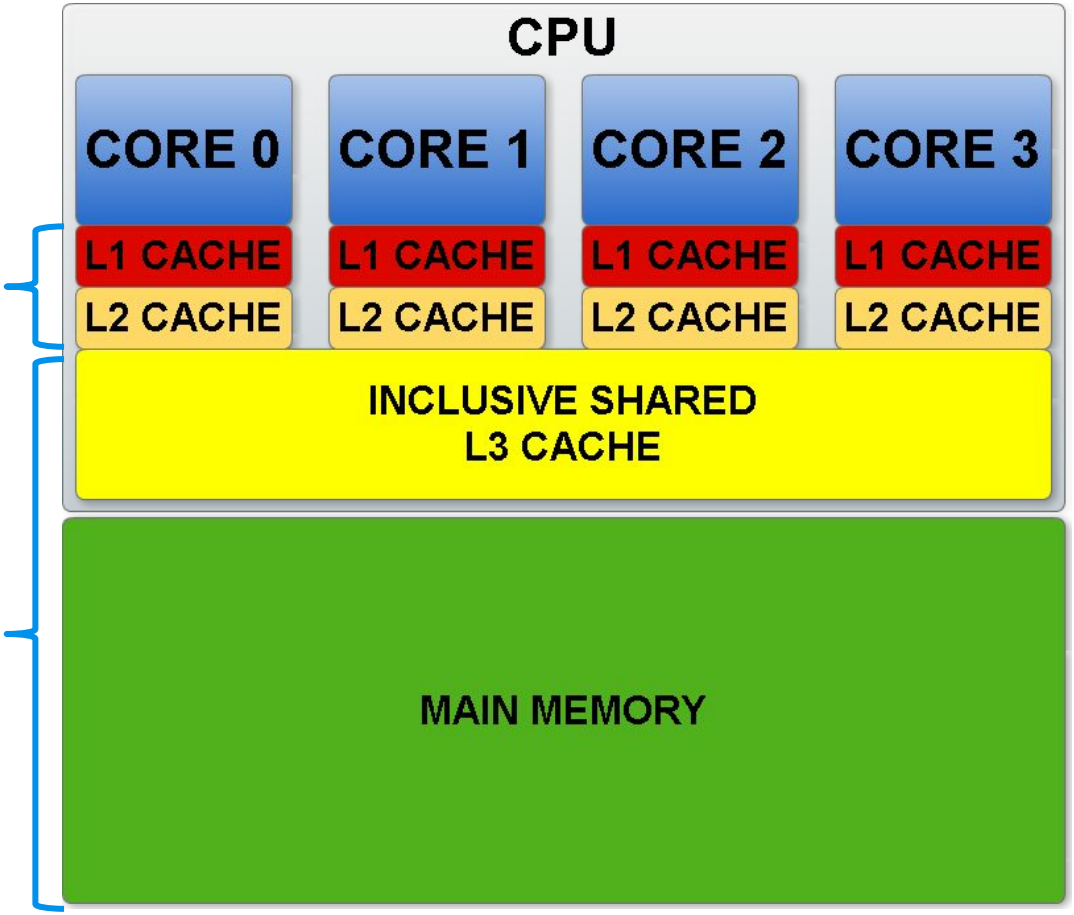
- Input graph is **read-only** in the shared memory
- Each algorithm execution has small footprint (read/write) that fits in the **private L1/L2 cache**

Many instances can run in parallel without write conflicts and cache coherence issues

Cache coherence:

"Working" memory

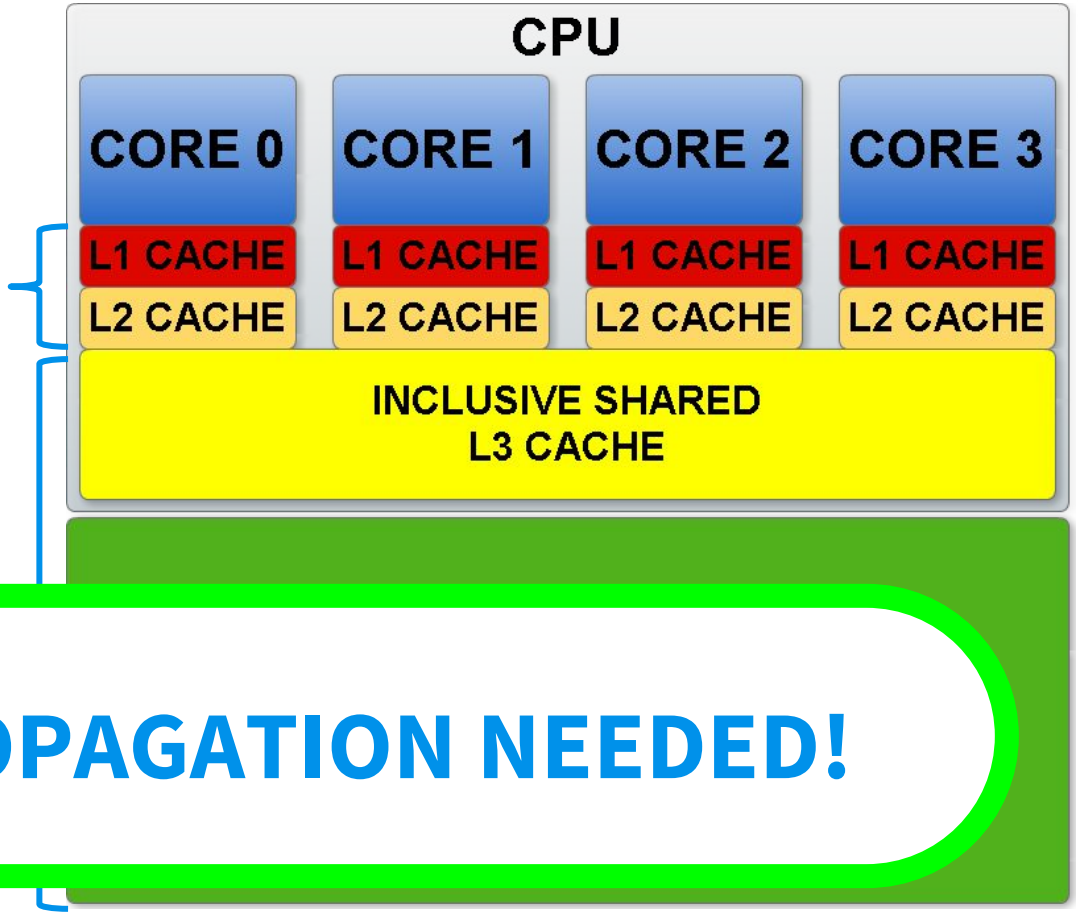
Input graph



Cache coherence:

"Working" memory

Input graph



NO PROPAGATION NEEDED!

Back to maximal cliques: state of the art

State of the art:

Algorithms can be classified as

- **Based on Bron-Kerbosch (BK)**
- **Based on Reverse-Search**

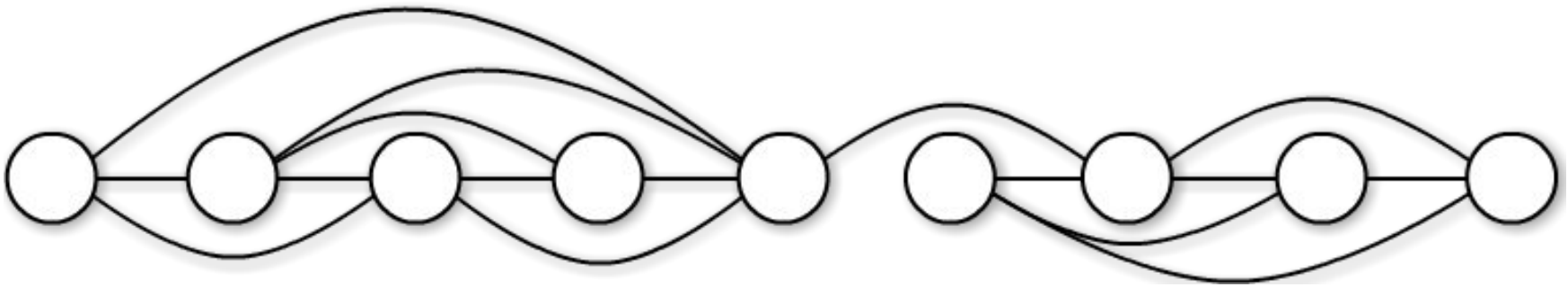
State of the art: BK-based

[**Eppstein *et al***] Listing All Maximal Cliques in Large Sparse Real-World Graphs *JEA 2013*

- Total running time: $\mathbf{O(nd3^{(d/3)})}$ (**d** is the **degeneracy**) which can be optimal on sparse graphs
- Additional space is $\mathbf{O(n + d\Delta)}$
- **Delay** is $\mathbf{\Omega(3^{n/6})}$

Degeneracy ordering:

- Each vertex has at most **d** neighbors to its right
- E.G., Recursively remove smallest-degree node



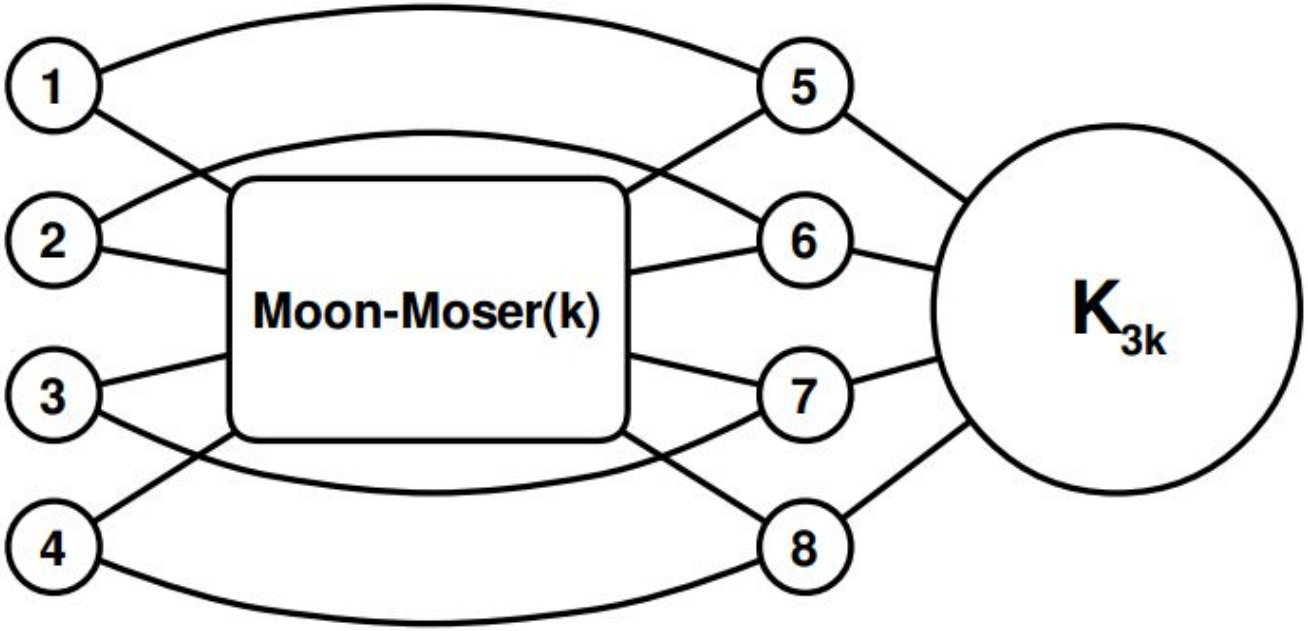
Example with **d** = 3

Eppstein *et al*:

Main idea

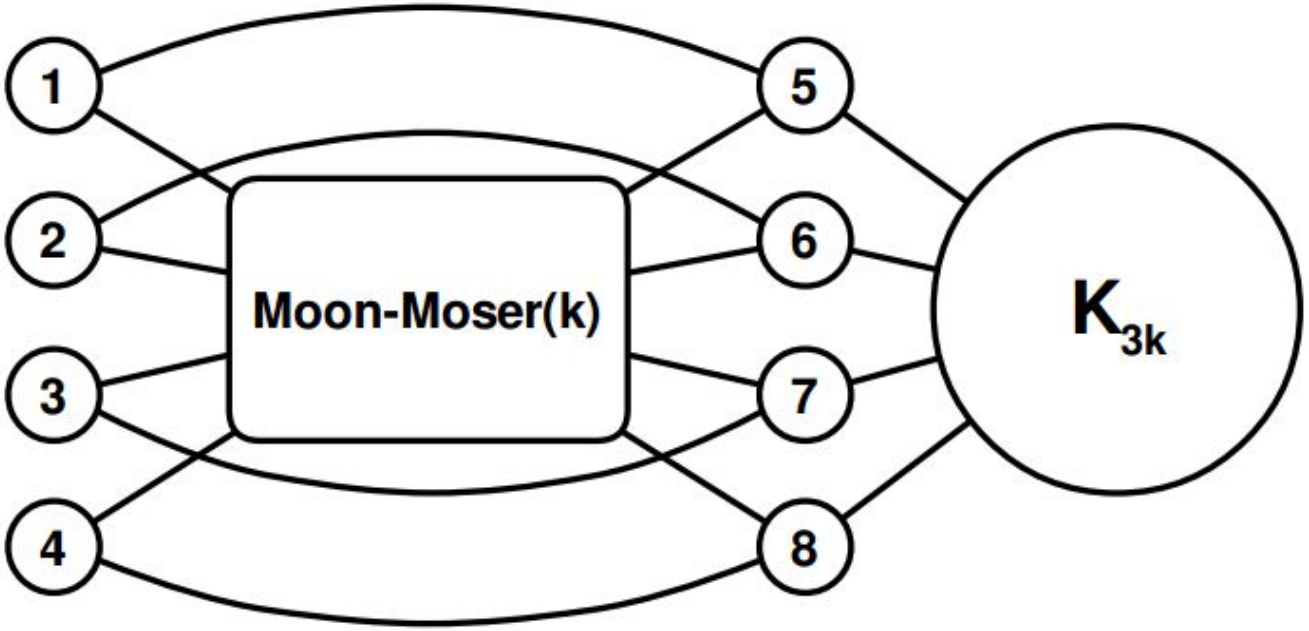
- Process the graph in a ***degeneracy*** *ordering*
- This creates **n** subgraphs with **d** nodes to visit

Eppstein *et al*: exponential delay



Degeneracy Ordering: **{1,2,3,4, x ...}**

Eppstein *et al*: exponential delay

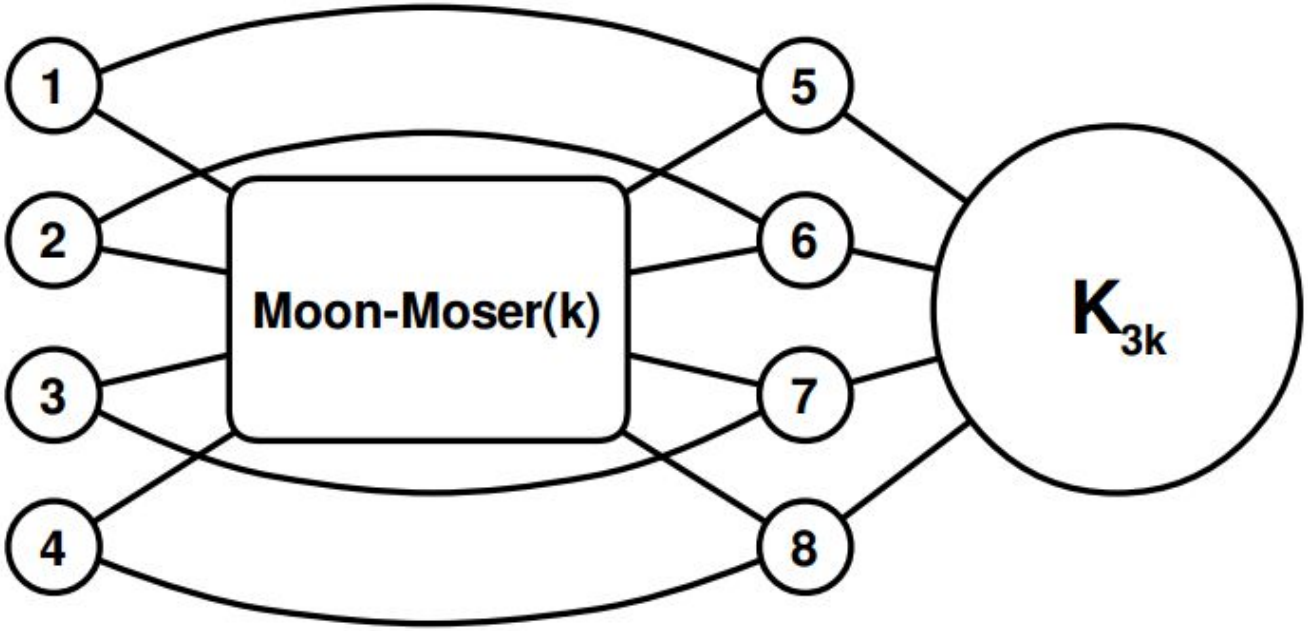


Degeneracy Ordering: **{1,2,3,4, x ...}**

(**x** is a node of the Moon-Moser graph)

When **x** is processed, NO clique is found!

Eppstein *et al*: exponential delay



Processing **x**: recursive calls make a *complete ternary tree* of height **k-1**, i.e. there are $\Theta(3^k) = \Theta(3^{n/6})$ calls

State of the art: reverse-search

[**Tsukiyama *et al***] A new algorithm for generating all the maximal independent sets

SIAM J COMPUT 1977

- Can be used for cliques (use the complement graph)
- Builds a solution tree which can be traversed
- Additional space is **O(m)**
- **Delay** is **O(mn)**

State of the art: reverse-search

- [Chiba-Nishizeki] $O(md)$ $O(m)$
- [Makino-Uno] $O(\Delta^4)$ $O(m)$
- [Chang *et al*] $O(\Delta h^3)$ $O(n^2)$
- [Comin-Rizzi] $O(n^{2.09})$ $O(n^{4.27})$
- **See paper for more…**

State of the art: reverse-search

- [Chiba-Nishizeki] $O(md)$ $O(m)$
- [Makino-Uno] $O(\Delta^4)$ $O(m)$
- [Chang *et al*] $O(\Delta h^3)$ $O(n^2)$
- [Comin-Rizzi] $O(n^{2.09})$ $O(n^{4.27})$
- See paper for more…
- [This work] $\tilde{O}(qd(\Delta+qd))$ $O(q)$
- [This work] $\tilde{O}(\min\{qd\Delta,md\})$ $O(d)$

sublinear!

Note: $q < d < h < \Delta < n$

Performance:

	DBLP-2008			AMAZON-0505			IN-2004			EU-2005		
n	511 163			410 236			1 353 703			862 664		
m	1 871 070			2 439 436			13 126 172			16 138 468		
α	447 563			1 034 135			3 384 922			5 727 256		
Δ	576			2 760			21 869			68 963		
d	114			10			488			388		
q	115			11			489			387		
ALGO.	TIME *sec*	DELAY *ms*	MEM *MiB*	TIME *sec*	DELAY *ms*	MEM *MiB*	TIME *sec*	DELAY *ms*	MEM *MiB*	TIME *sec*	DELAY *ms*	MEM *MiB*
CN	>2h	475	97.56	>2h	636	78.26	>2h	5 285	258.21	>2h	4 077	164.54
MU	39.5	18	3.01	16.9	22	3.01	6 102.7	3 691	52.62	>2h	11 395	30.52
CXQ	21.2	4	0.11	60.7	12	0.25	>2h	6 004	1.02	>2h	621	3.15
RALG2	1.8	0.6	1.57	3.1	0.8	0.81	66.6	401	6.46	237.4	245	3.16
ALG2	2.3	15	0.01	4.4	0.9	0.01	100.7	410	0.03	363.8	277	0.02

CN: Chiba-Nishizeki [11] MU: Sparse graph Makino-Uno [25] CXQ: Chang et al. [9]
RALG2: Recursive Algorithm 2 ALG2: Algorithm 2

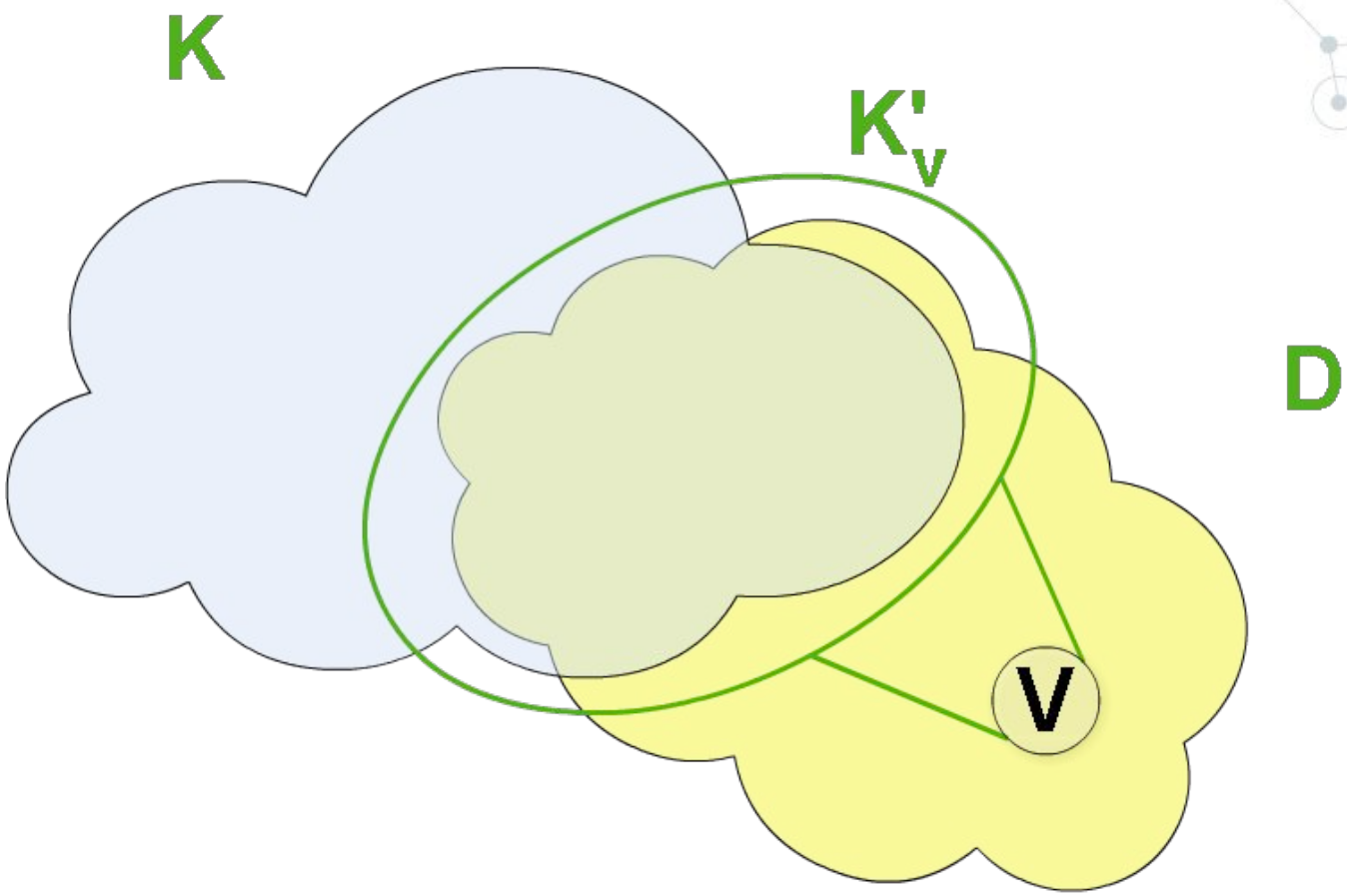
Comparison:

	DBLP-2008			AMAZON-0505			IN-2004			EU-2005		
n	511 163			410 236			1 353 703			862 664		
m	1 871 070			2 439 436			13 126 172			16 138 468		
α	447 563			1 034 135			3 384 922			5 727 256		
Δ	576			2 760			21 869			68 963		
d	114			10			488			388		
q	115			11			489			387		
ALGO.	TIME *sec*	DELAY *ms*	MEM *MiB*	TIME *sec*	DELAY *ms*	MEM *MiB*	TIME *sec*	DELAY *ms*	MEM *MiB*	TIME *sec*	DELAY *ms*	MEM *MiB*
CN	>2h	475	97.56	>2h	636	78.26	>2h	5 285	258.21	>2h	4 077	164.54
MU	39.5	18	3.01	16.9	22	3.01	6 102.7	3 691	52.62	>2h	11 395	30.52
CXQ	21.2	4	0.11	60.7	12	0.25	>2h	6 004	1.02	>2h	621	3.15
RALG2	1.8	0.6	1.57	3.1	0.8	0.81	66.6	401	6.46	237.4	245	3.16
ALG2	2.3	15	0.01	4.4	0.9	0.01	100.7	410	0.03	363.8	277	0.02

CN: Chiba-Nishizeki [11] MU: Sparse graph Makino-Uno [25] CXQ: Chang et al. [9]
RALG2: Recursive Algorithm 2 ALG2: Algorithm 2

Our algorithm

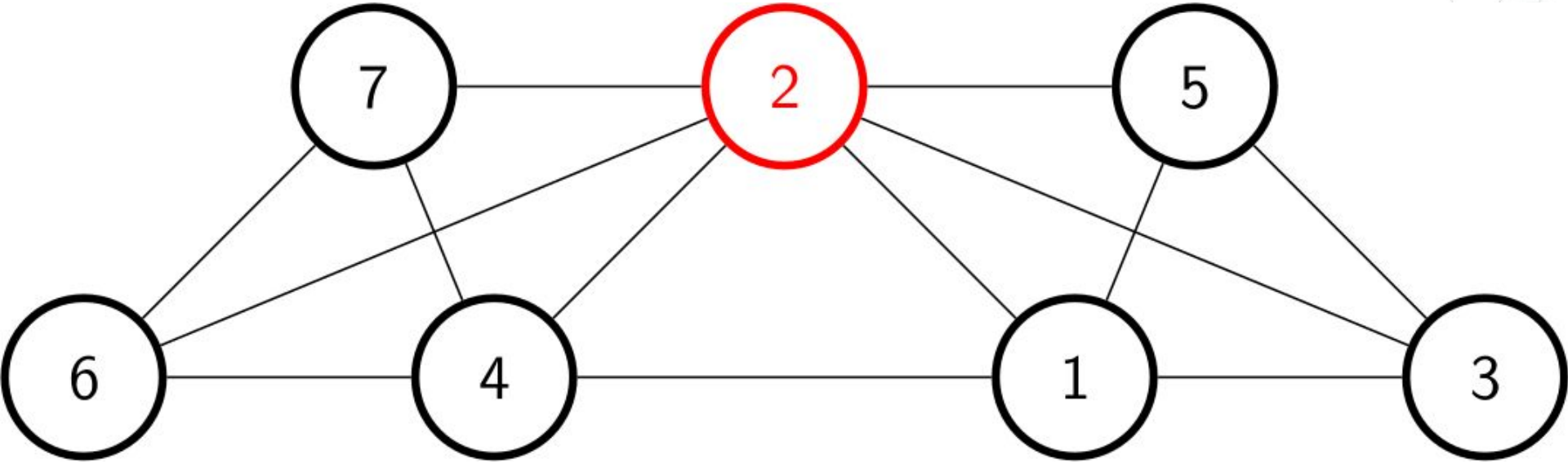
Child clique:



D = COMPLETE(K'U{V})

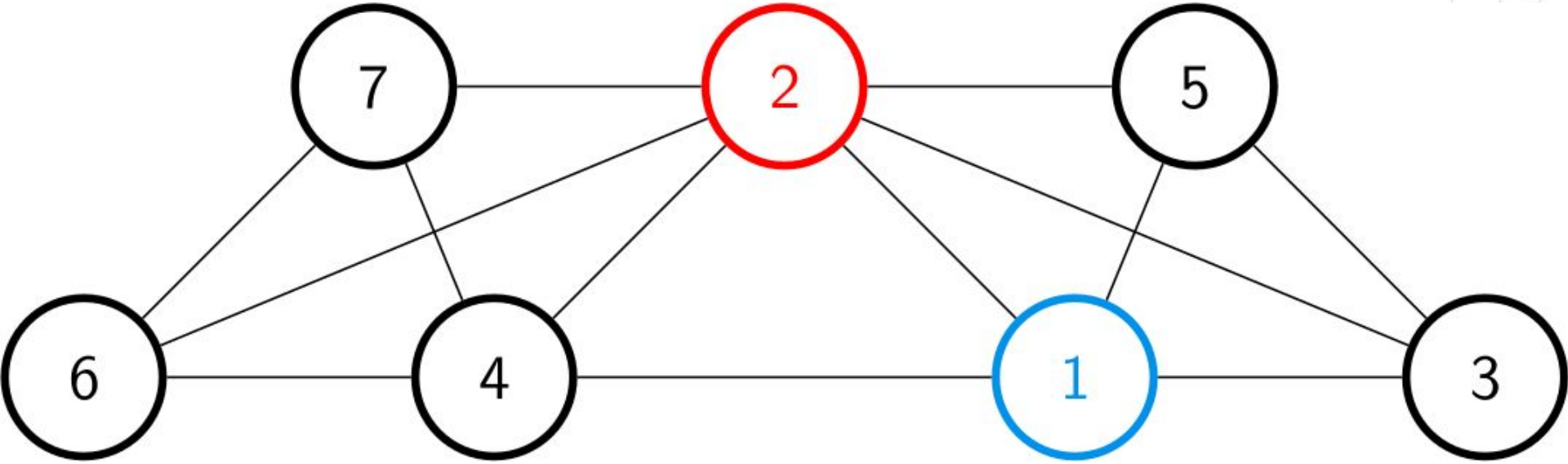
Complete:

COMPLETE({2})



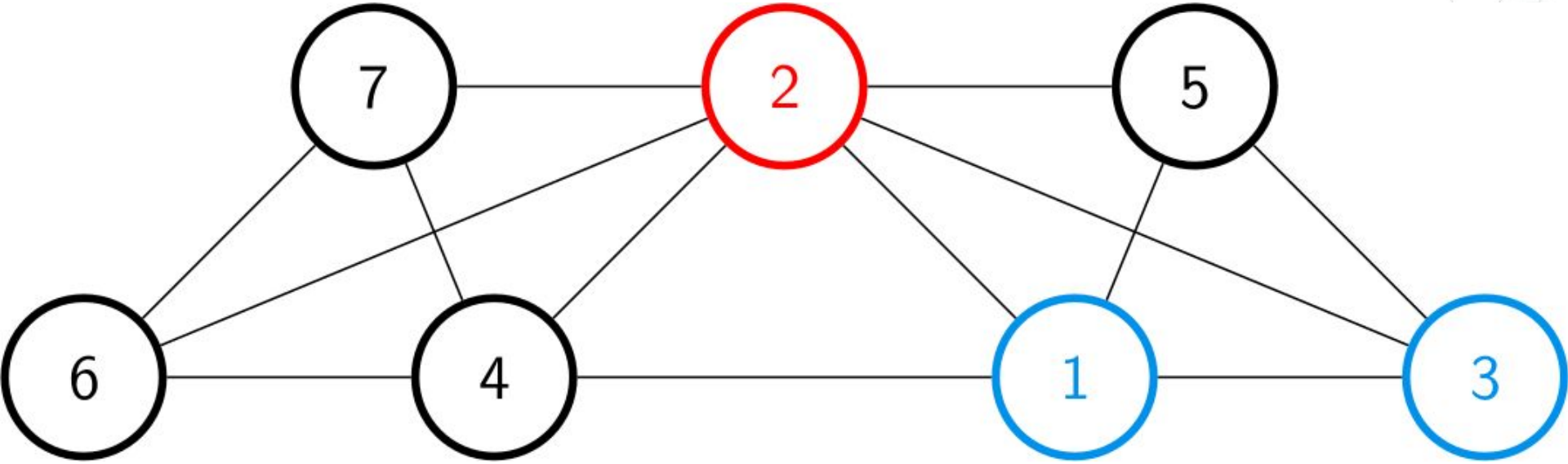
Complete:

COMPLETE({2})



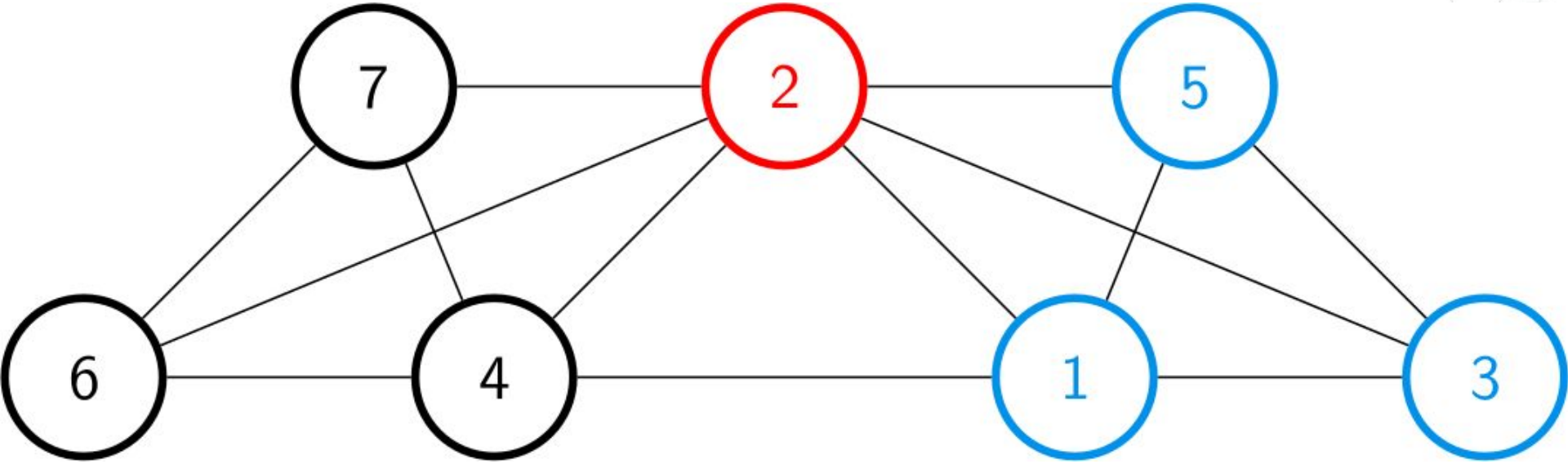
Complete:

COMPLETE({2})



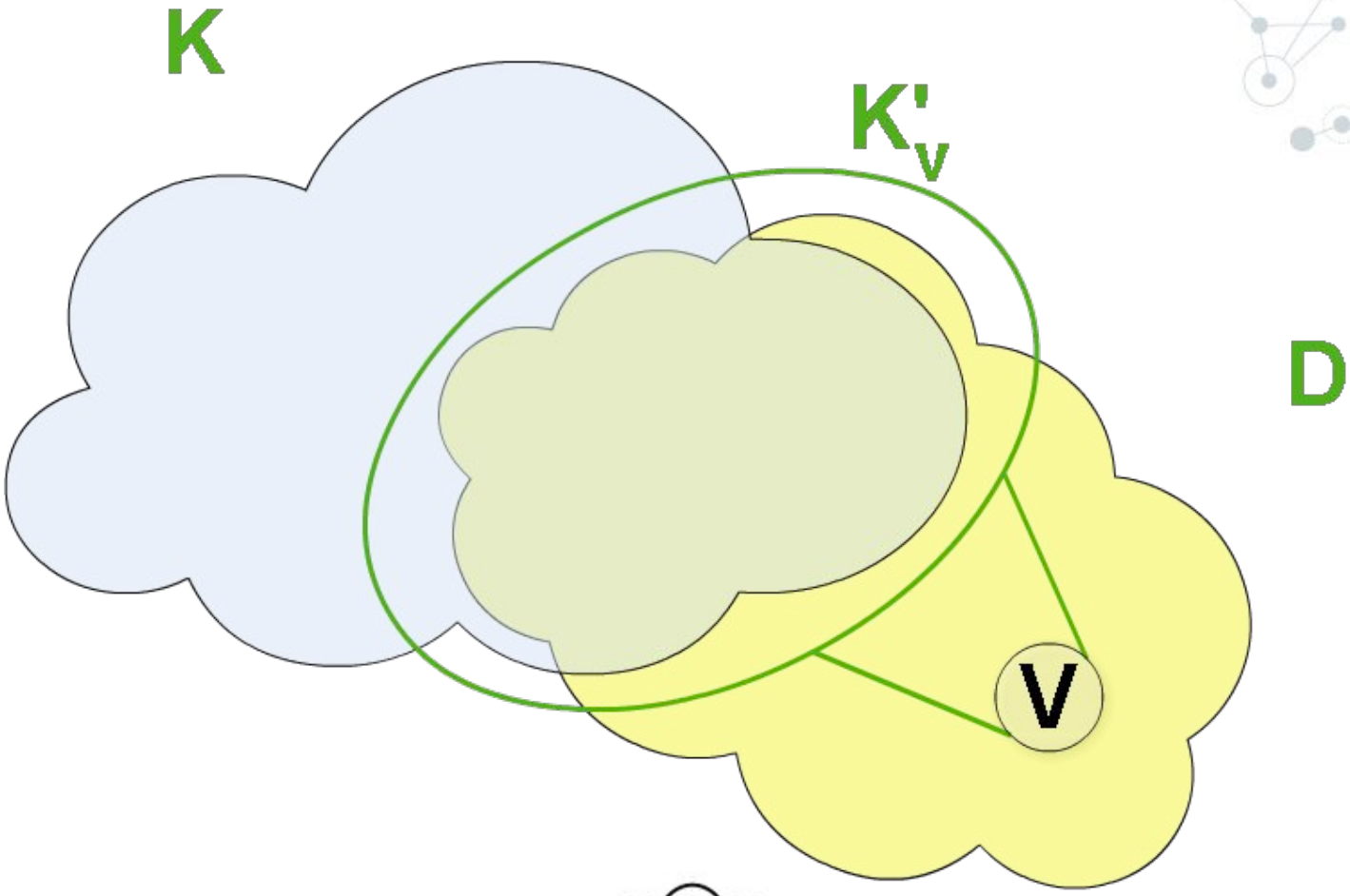
Complete:

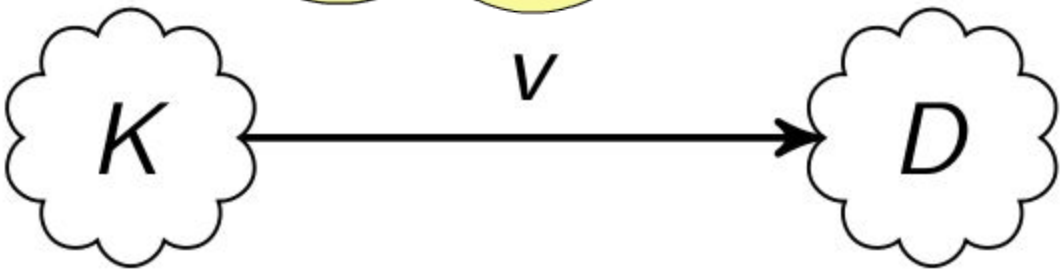
COMPLETE({2})



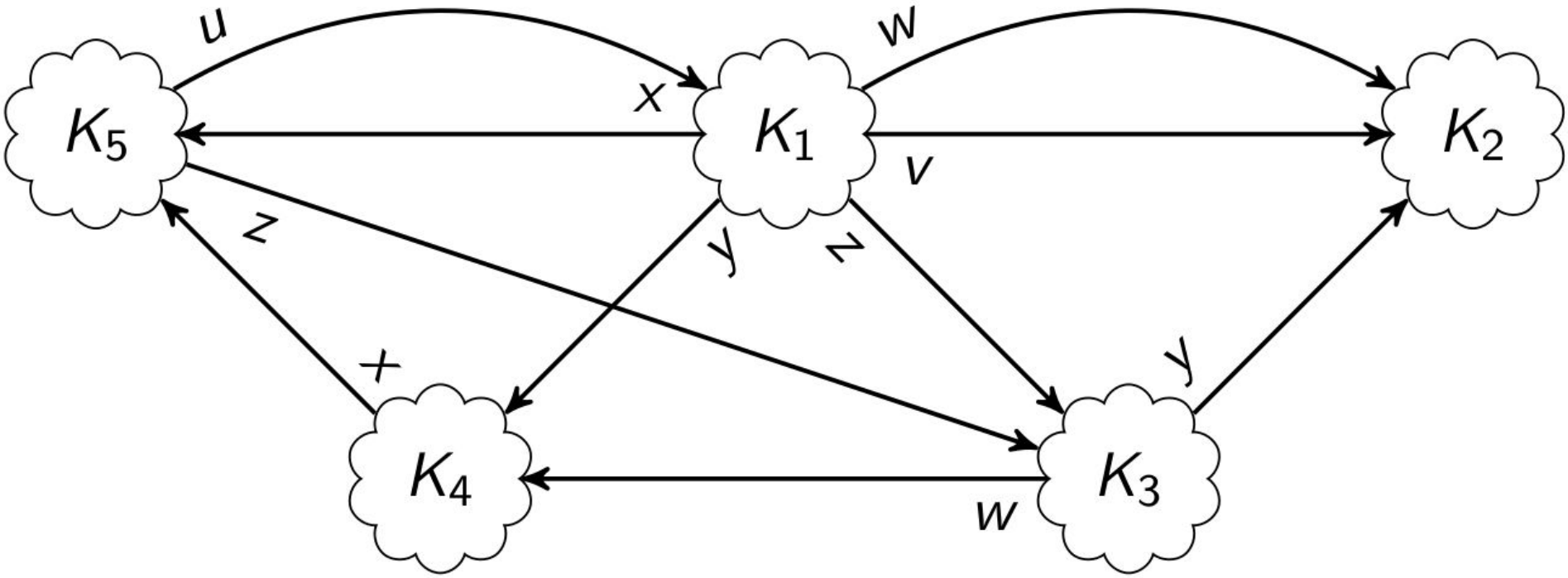
MAXIMAL!

Child clique:

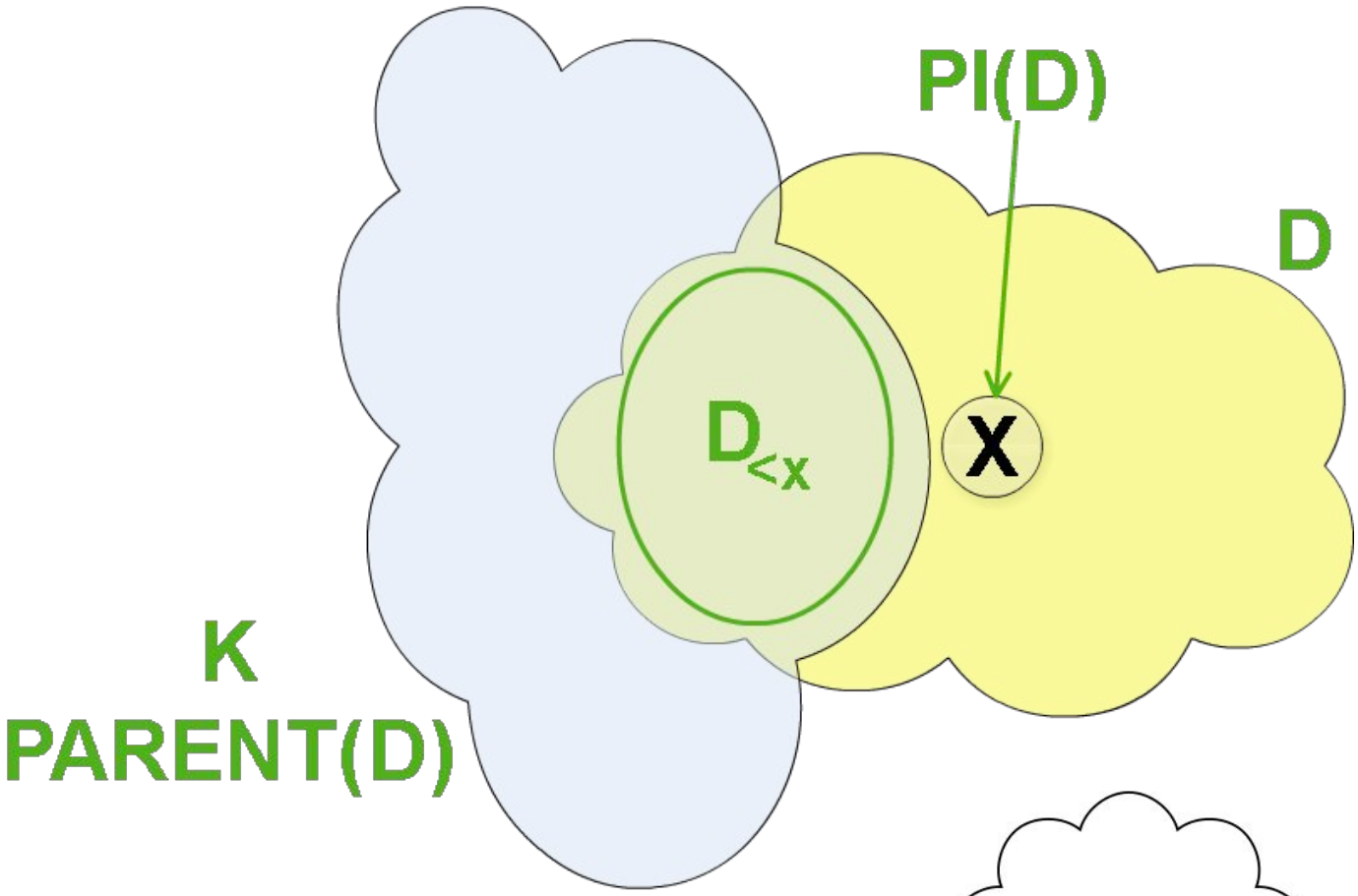


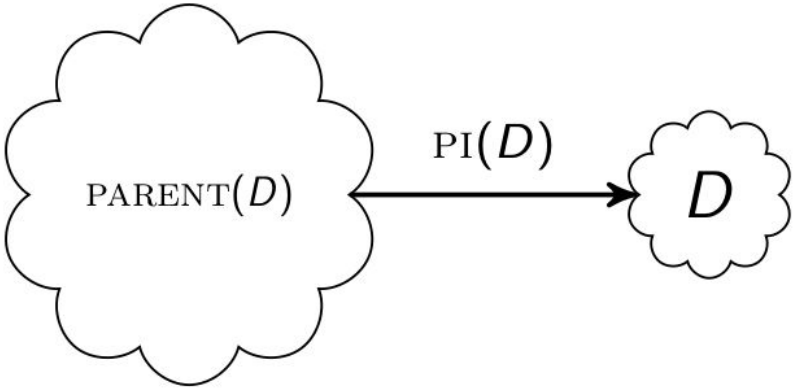


Solution graph:

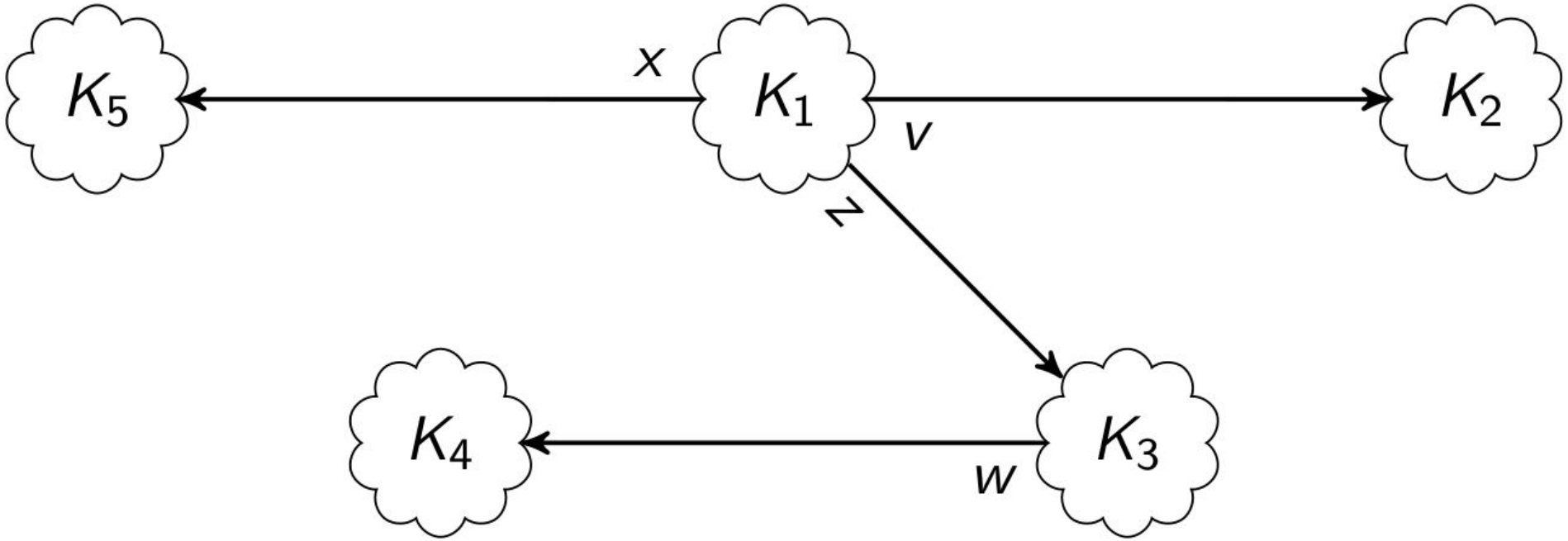


Parent clique:





Implicit solution tree:



State during the implicit tree traversal:
current node + current edge.
Going up from K ⇔ parent(K) and PI(K).

Checking a child [Makino-Uno]:

Check if an edge should be used (the straight way):

PARENT(D) = K and PI(D) = v

THIS IS THE COMPUTATIONAL BOTTLENECK!!

Improved Check:

From:

PARENT(D) = K and PI(D) = v

To (our way):

$N(K'_v)$ is disjoint from $N_<(v)$ and B_K

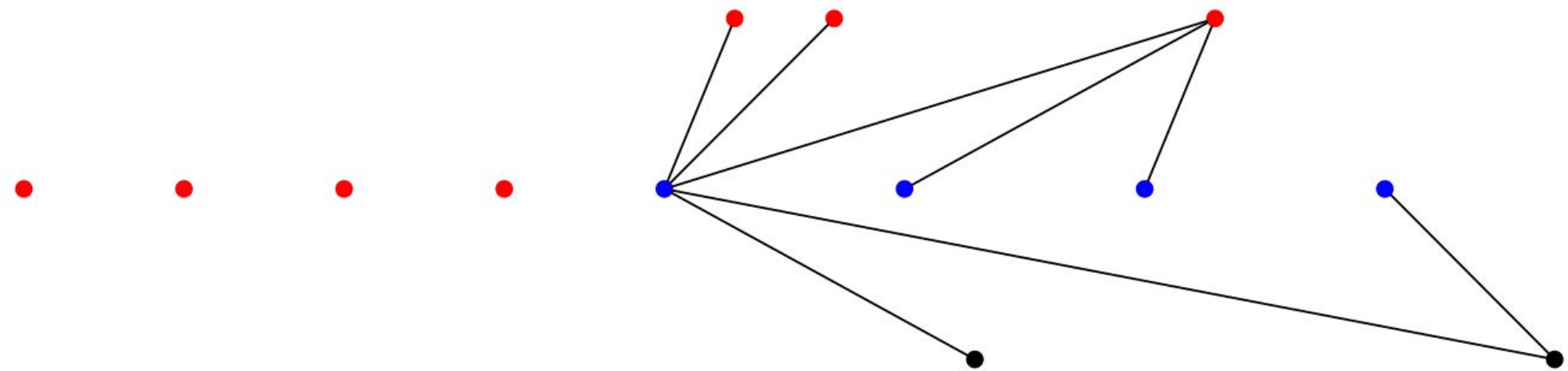
The children of a clique can now be found in

$O(qd\Delta)$!

What is B_K?

v is in B_K if it satisfies the following property:

For each $u \in K$ smaller than v, u is a neighbor of v



K **B_K**

Improved delay:

Compute B_K every time we change clique and **store** it:

O(d) space

O(min(qdΔ,md)) time

O(q) space:

Iterate over B_K in increasing order whenever we need it:

O(q) space

O(qd(Δ+qd)) time

Conclusions:

- Output-sensitive algorithms can be **efficient** both in theory and in practice
- Stateless search can heavily reduce **space usage**
- We showed these principles applied to **MCE**

The end.

Thank you for your attention.

Any questions?