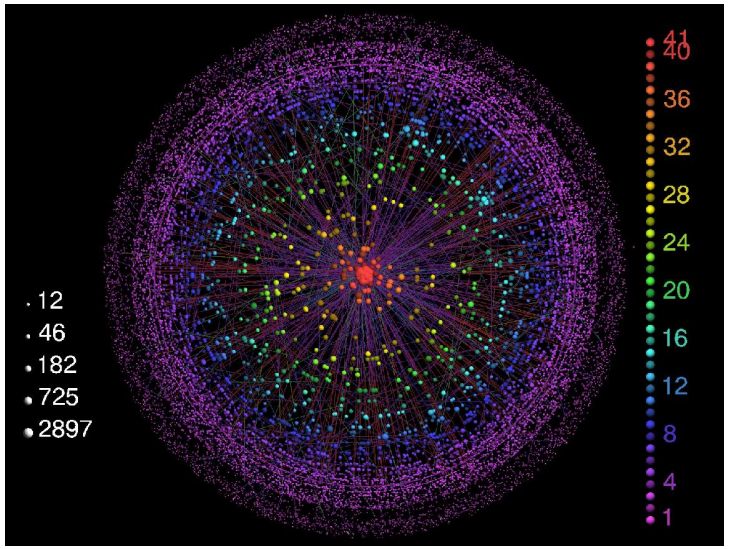
Hierarchical Core Decomposition in Parallel: From Construction to Subgraph Search

Deming Chu, Fan Zhang, Wenjie Zhang, Xuemin Lin, Ying Zhang

Core Decomposition

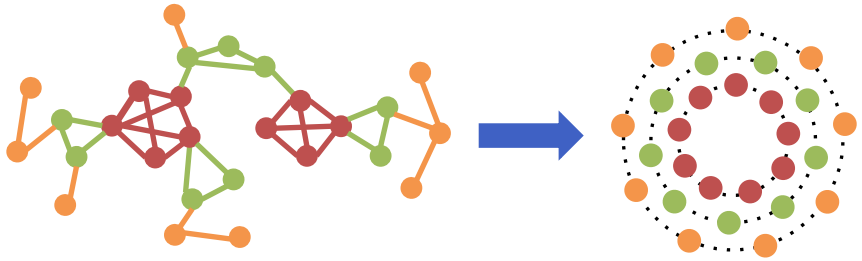
- Powerful Tool in Network Analysis
- Decompose a graph into layers



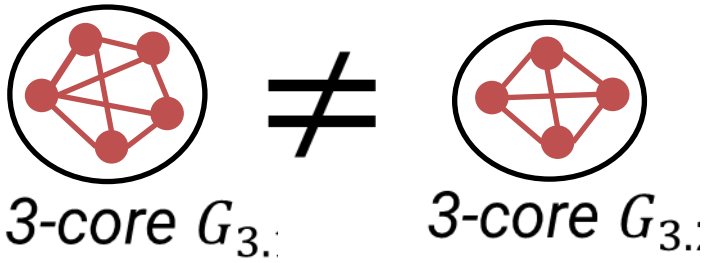
UNSW SYDNEY

Limitations of Core Decomposition

1. **Connectivity of k-cores is lost**
2. **Containment of k-cores is lost**



1.



2.

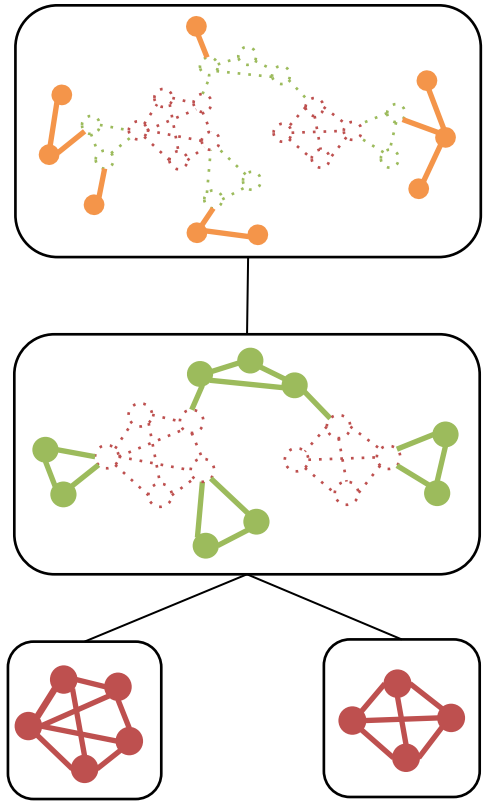
UNSW SYDNEY

1. Hierarchical Core Decomposition (HCD)

- Connectivity and Containment of Different k-cores
- Stored in $O(n)$ space, $\boldsymbol{n}$: #vertices

P-Complete proof (inherently sequential)

Near-linear work parallel algorithm



Applications of HCD

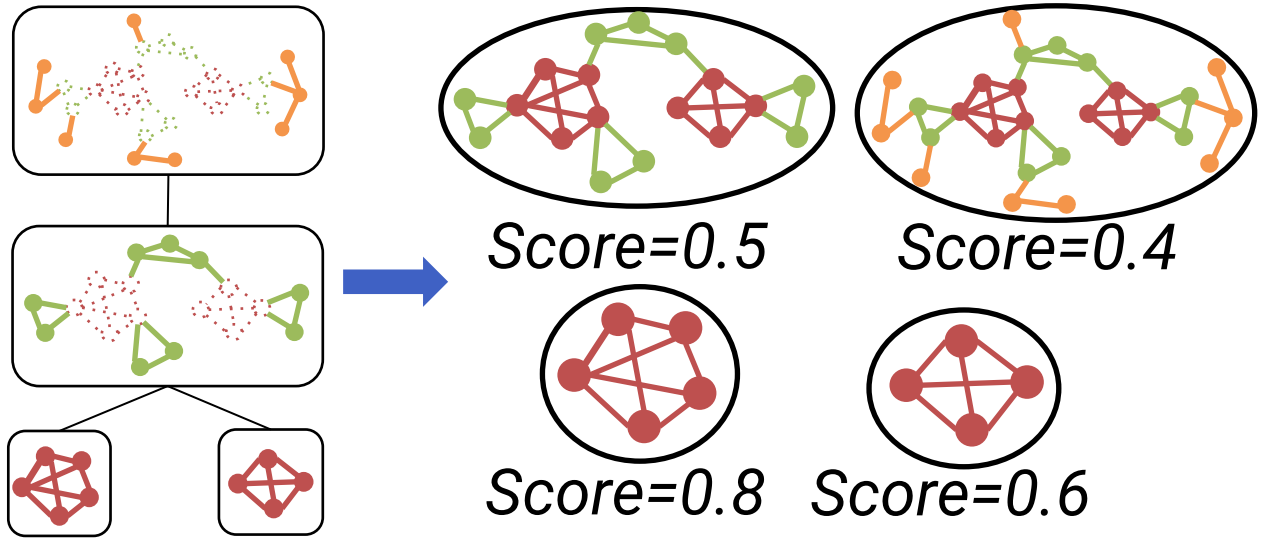
- **Cohesive Subgraph Search**
  - Find influential k-core, attributed k-core, the densest subgraph
- **User Engagement Analysis**
  - Gives more accurate prediction than core decomposition
- **Graph Visualization**
  - Visualize the internet, biology, and brain networks

2. Subgraph Search on HCD

- Find high-quality subgraphs on HCD
- Community scoring metric to evaluate subgraph quality
  - E.g. average degree, clustering coefficient

Work-Efficient Parallel Algorithms

Work-Efficient: *#steps matches the best serial time complexity*



Applications of Subgraph Search on HCD

- **Densest Subgraph**
  - Our solution is the STOA approximate solution for densest subgraph
- **Maximum Clique**
  - Our solution can be a potential pruning strategy for maximum clique
- **Size-Constrained k-Core [1]**

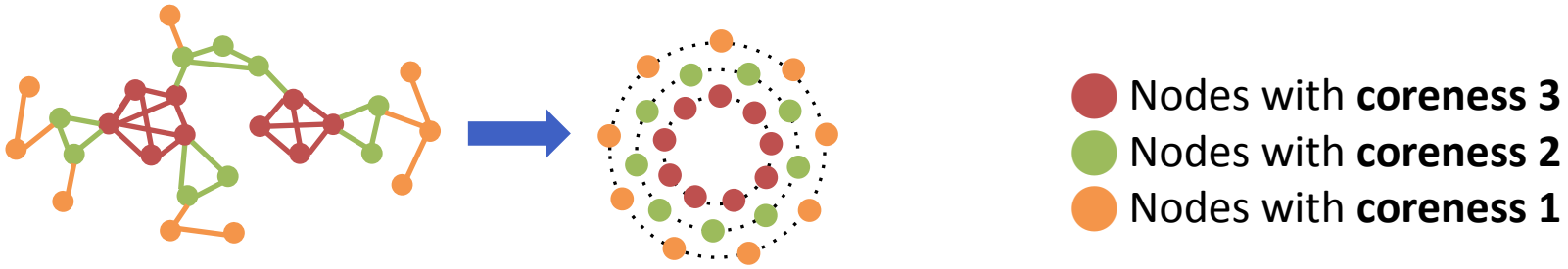
[1] D. Chu, F. Zhang, X. Lin, W. Zhang, Y. Zhang, Y. Xia, and C. Zhang, “Finding the best k in core decomposition: A time and space optimal solution,” in ICDE. IEEE, 2020, pp. 685–696.

Roadmap

- ✔ Problem Definition
  - HCD Construction
  - Subgraph Search on HCD
- Parallel Construction of HCD
- Parallel Subgraph Search on HCD
- Experimental Results
- Conclusion

K-Core and Core Decomposition

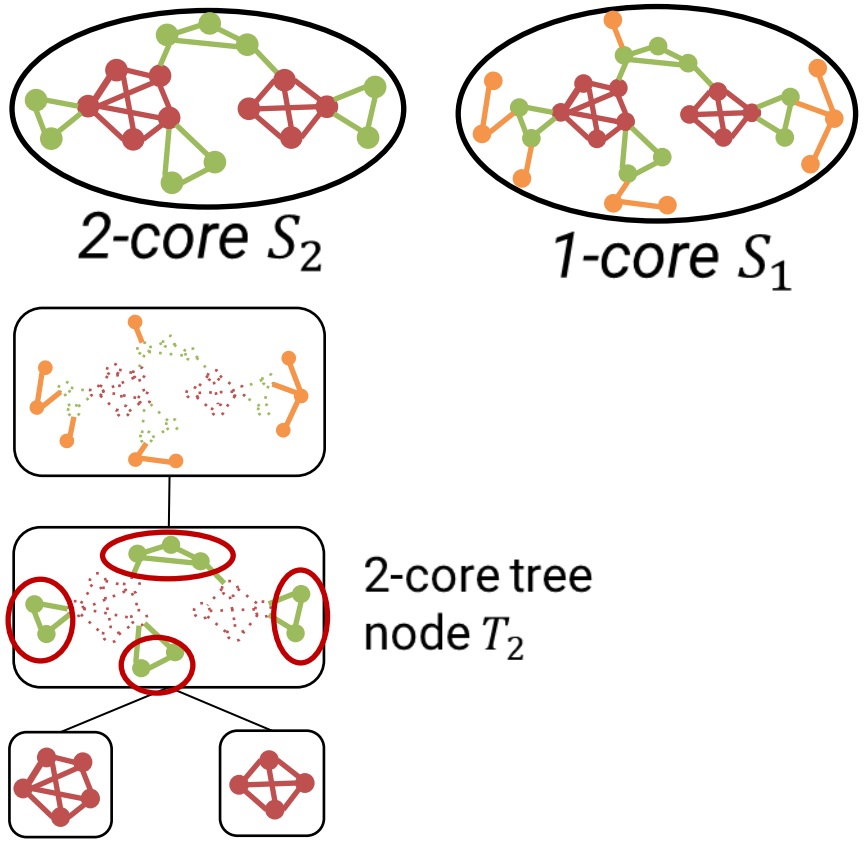
- ***k*-Core**: the maximal **connected** subgraph where each node has at least k neighbors
- ***Coreness***: largest k such that the node is in the k-core



1. Hierarchical Core Decomposition

- ***k*-Core Tree Node:** T_i stores the vertices with coreness k in a k-core S_i
- ***Parent Tree Node*:** k_1-core tree node T_1 is the parent of k_2-core tree node T_2 if
  1. $k_1 < k_2$
  2. $S_2 \subset S_1$
  3. No k'-core S', $S_2 \subset S' \subset S_1$

- ***HCD***: find all k-core tree nodes and their parent tree node



Definition: Community Scoring Metric

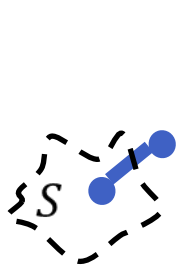
Primary Value (for a subgraph S)

Most metrics are based on Primary Values

$n(S)$: # of nodes

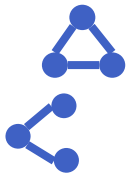
$m(S)$: # of edges

$b(S)$: # of boundary edges



$\Delta(S)$: # of triangles

$t(S)$: # of triplets



Higher-Order

Community Scoring Metric (for a subgraph S)

To measure community quality

Type-A

Average Degree: $\frac{2\times m(S)}{n(S)}$

Internal Density: $\frac{2\times m(S)}{n(S)\times(n(S)-1)}$

Cut Ratio: $1-\frac{b(S)}{n(S)\times(n-n(S))}$

Modularity: $\sum_{i=1}^{k}\left(\frac{m(P_i)}{m}-\left(\frac{2\times m(P_i)+b(P_i)}{2\times m}\right)^2\right)$

Conductance: $1-\frac{b(S)}{2\times m(S)+b(S)}$

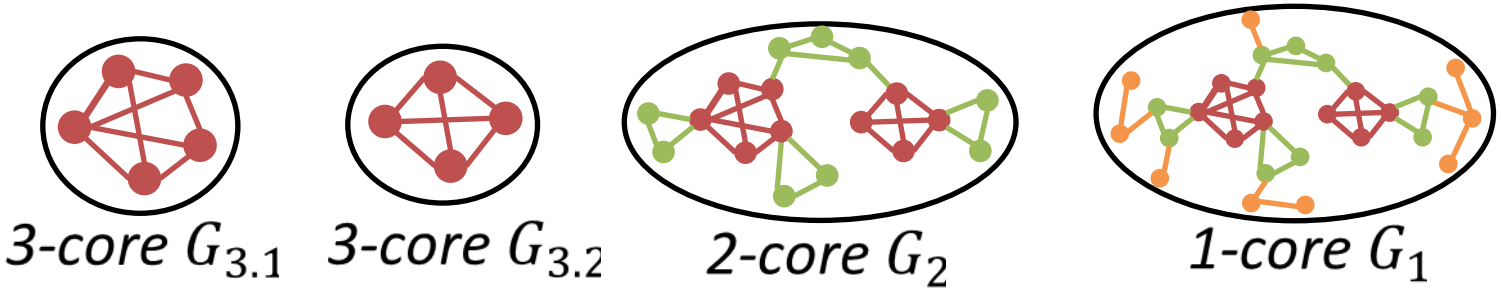
Type-B

Clustering Coefficient: $\frac{3\times\Delta(S)}{t(S)}$

2. Subgraph Search on HCD

Given: Graph G, Community Scoring Metric Q

Subgraph Search: find the k-core with the highest score w.r.t. Q among all k-cores



Roadmap

- Problem Definition
- ✔ Parallel Construction of HCD
- Parallel Subgraph Search on HCD
- Experimental Results
- Conclusion

Existing Works of HCD Construction

- LCPS [1]: SOTA serial algorithm, $O(m)$, $\boldsymbol{m}$: #edges
  Concurrent visits of vertices -> inconsistent priority orderings and results
- Other serial methods are much slower

Existing works are hard to parallelize

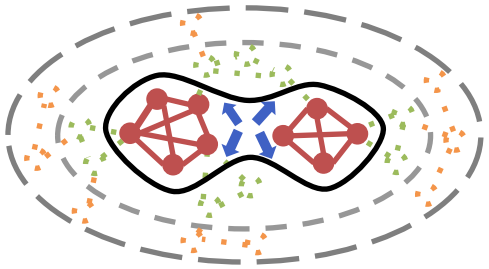
Divide & Conquer method is infeasible (in experiments)

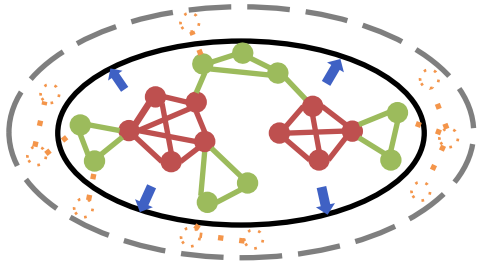
Design a new union-find-based solution

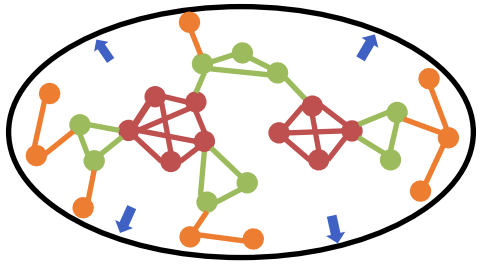
[1] D. W. Matula and L. L. Beck, “Smallest-last ordering and clustering and graph coloring algorithms,” J. ACM, vol. 30, no. 3, pp. 417–427, 1983.

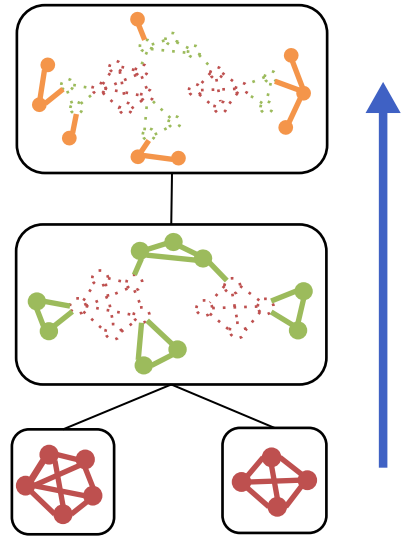
Intuition of Our Solution

- Add vertices to union-find in decreasing coreness
- Meanwhile, build HCD from bottom to up
- Use pivot to identify tree node and its parent







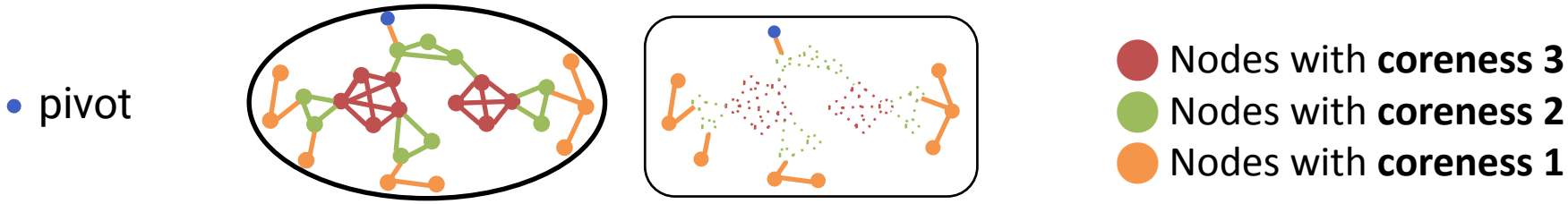


Vertex Rank and Pivot

- ***Vertex Rank*: sort nodes by coreness, and break tie by id**
- ***Pivot*:** the vertex with the lowest vertex rank in a subgraph

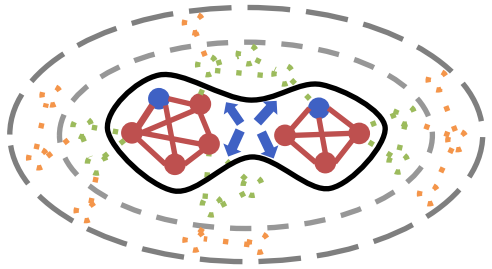
Pivot can uniquely identify k-core and its tree node

1. The pivot is unique
2. The pivot of the k-core is in the tree node



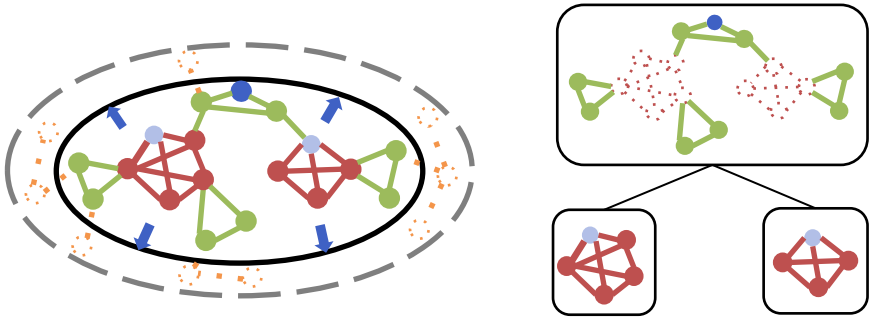
Vertex Rank and Pivot

1. Group Vertices



- pivot

2. Find Parent



- pivot
- old pivot

Our Solution

for k from k_{max} down to 0:

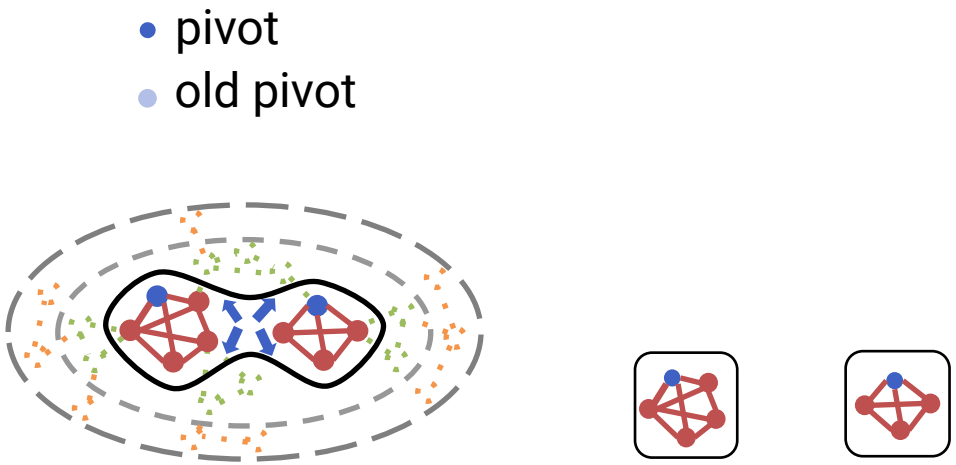
1. Find k'-Core Tree Node ($k' > k$)
2. Union in union-find
3. Create Tree Nodes
4. Find Parent Tree Node

Our Solution

for k from k_{max} down to 0:

1. Find k'-Core Tree Node $(k' > k)$
2. Union in union-find
3. Create Tree Nodes
4. Find Parent Tree Node

add ● to the graph

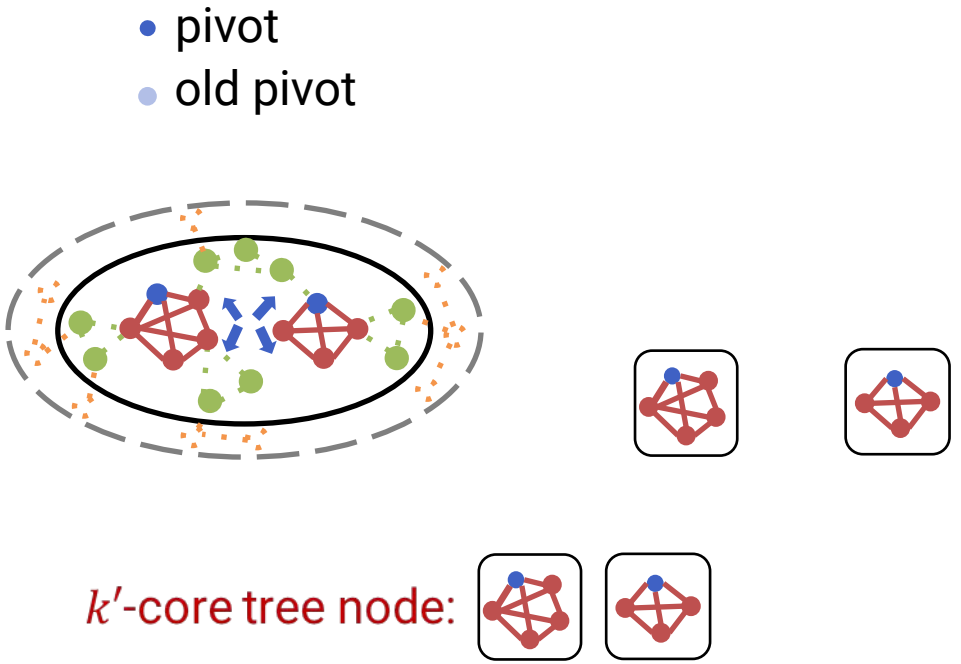


Our Solution

for k from k_{max} down to 0:

1. Find k'-Core Tree Node ($k' > k$)
2. Union in union-find
3. Create Tree Nodes
4. Find Parent Tree Node

1. Visit ● 's neighbors, identify the k'-core that will union with ●

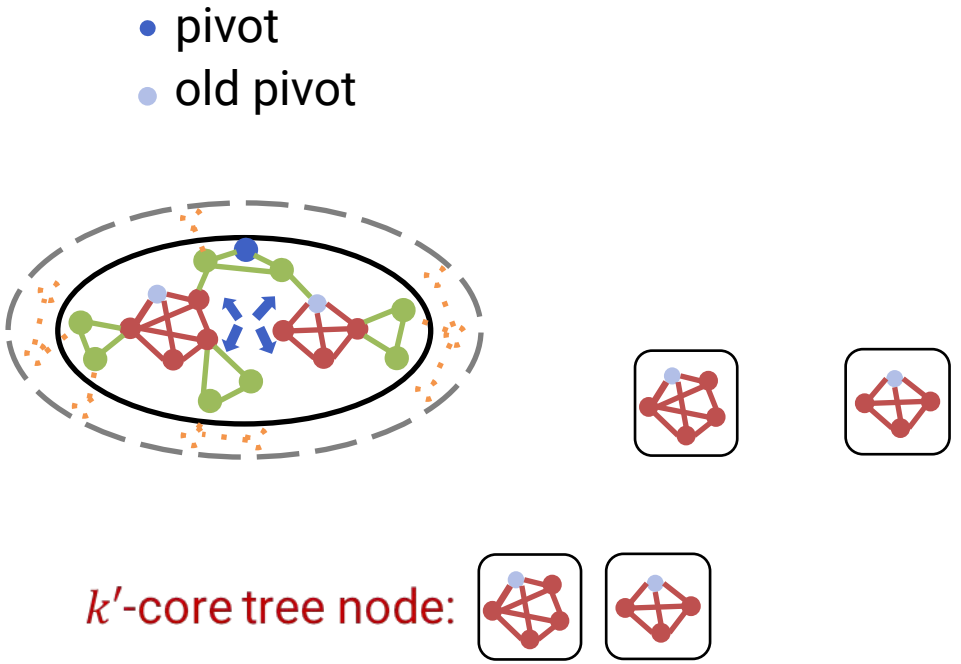


Our Solution

for k from k_{max} down to 0:

1. Find k'-Core Tree Node ($k' > k$)
2. Union in union-find
3. Create Tree Nodes
4. Find Parent Tree Node

2. Union ● with neighbors in union-find and pivot is changed

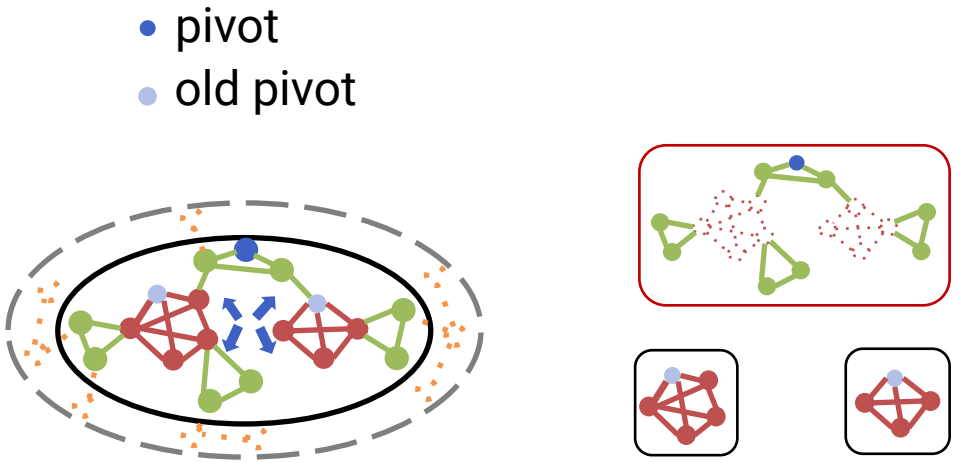


Our Solution

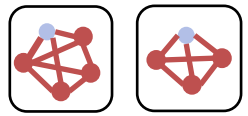
for k from k_{max} down to 0:

1. Find k'-Core Tree Node ($k' > k$)
2. Union in union-find
3. Create Tree Nodes
4. Find Parent Tree Node

3. Group ● by pivot into tree node



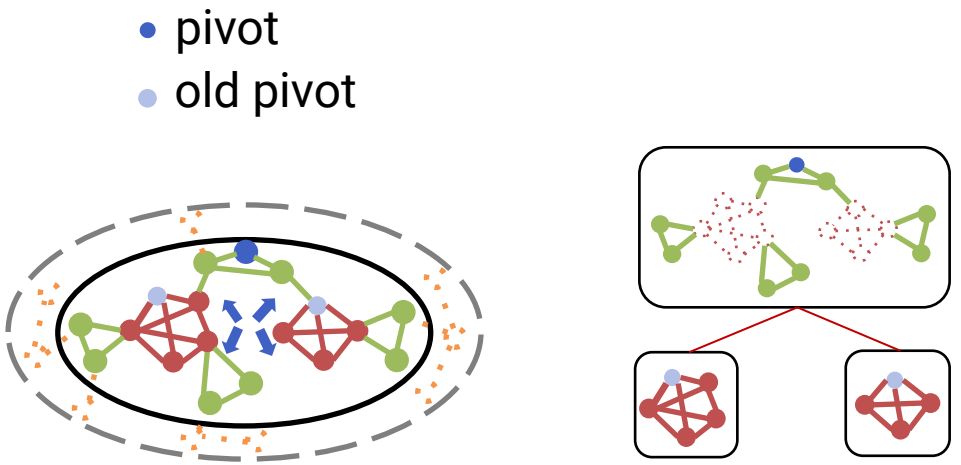
k'-core tree node:



Our Solution

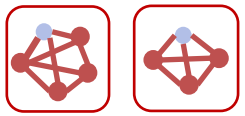
for k from k_{max} down to 0:

1. Find k'-Core Tree Node ($k' > k$)
2. Union in union-find
3. Create Tree Nodes
4. Find Parent Tree Node



4. In the same CC, the tree node containing ● and ● have parent-child relation

k'-core tree node:



Union-find with Pivot

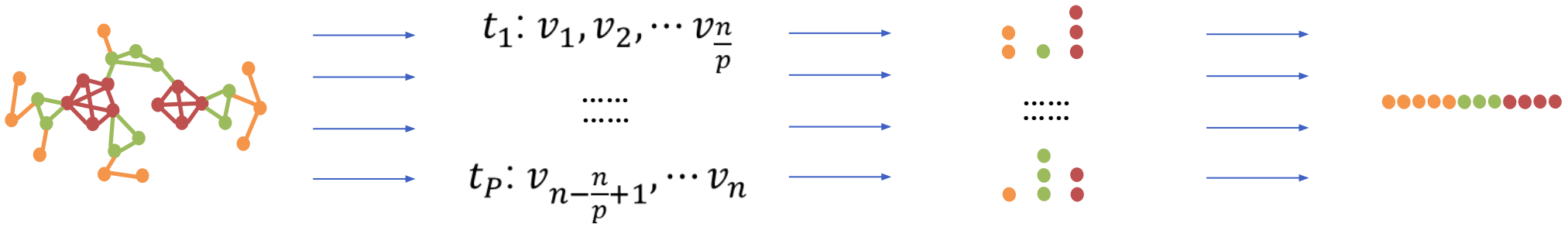
- Wait-free Union-Find (parallel)
- Maintain the pivot of connected component

- ***Time:*** $O\left(n\sqrt{p} + m\alpha(n) + F\right)$ work upon m operations/edges

$\boldsymbol{n}$: #vertices $\boldsymbol{p}$: #threads $\boldsymbol{F}$ failures in wait-free union-find
$\boldsymbol{\alpha(n)}$ is the reverse Ackermann function, $\alpha(n) \leq 4$ for any practical n

Parallel Vertex Rank Computation

- $O(n)$ work



$\boldsymbol{t_i}$: the i-th thread
$\boldsymbol{p}$: #threads

1. Distribute vertices

2. Group vertices by coreness

3. Place back into array, obtain vertex rank

⟶ is parallel

PHCD Analysis

- ***Hardness:*** we prove P-Completeness (inherently sequential)
- ***Space:*** $O(n)$, each vertex is in exactly one tree node
- ***Time:*** $O\left(n\sqrt{p} + m\alpha(n) + F\right)$
  - near-linear work, $m\alpha(n) \leq 4m$ for any practical n
  - uniting all edges in wait-free union-find

$\boldsymbol{n}$: #vertices $\boldsymbol{m}$: #edges $\boldsymbol{p}$: #threads $\boldsymbol{F}$ failures in wait-free union-find
$\boldsymbol{\alpha(n)}$ is the reverse Ackermann function, $\alpha(n) \leq 4$ for any practical n

Roadmap

- Problem Definition
- Parallel Construction of HCD
- ✔ Parallel Subgraph Search on HCD
- Experimental Results
- Conclusion

Existing Works of Subgraph Search

Limitations of BKS [1] (time- and space-optimal serial solution)

- Compute in decreasing coreness, rely on the results of larger coreness
- Preprocessing in BKS is inefficient to parallelize

Our Solution

- Vertex-centric solution (no dependency)
- A novel preprocessing
- Efficient to parallelize

[1] D. Chu, F. Zhang, X. Lin, W. Zhang, Y. Zhang, Y. Xia, and C. Zhang, “Finding the best k in core decomposition: A time and space optimal solution,” in ICDE. IEEE, 2020, pp. 685–696.

Our Solution

1. Compute HCD
2. Preprocessing
3. Score Computation on HCD
4. Output the Best K-Core

Our Solution

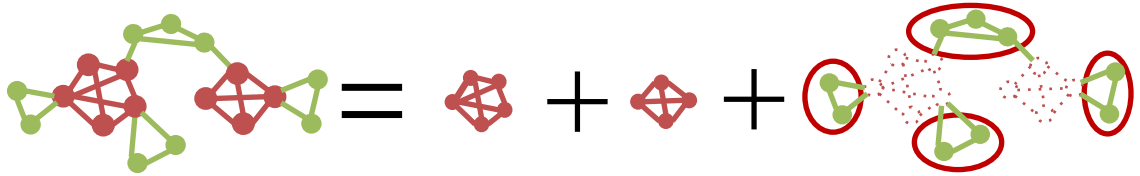
1. Compute HCD
2. Preprocessing
3. Score Computation on HCD
4. Output the Best K-Core

Preprocessing of every v:

- $gt(v)$: # neighbors $c(u) > c(v)$
- $eq(v)$: # neighbors $c(u) = c(v)$

The rest neighbors $c(u) < c(v)$

3. Score Computation on HCD (Intuition)



k-core = offsprings + k-core tree node

$f(k\text{-core}) = \sum f(\text{each son}) + \Delta f$

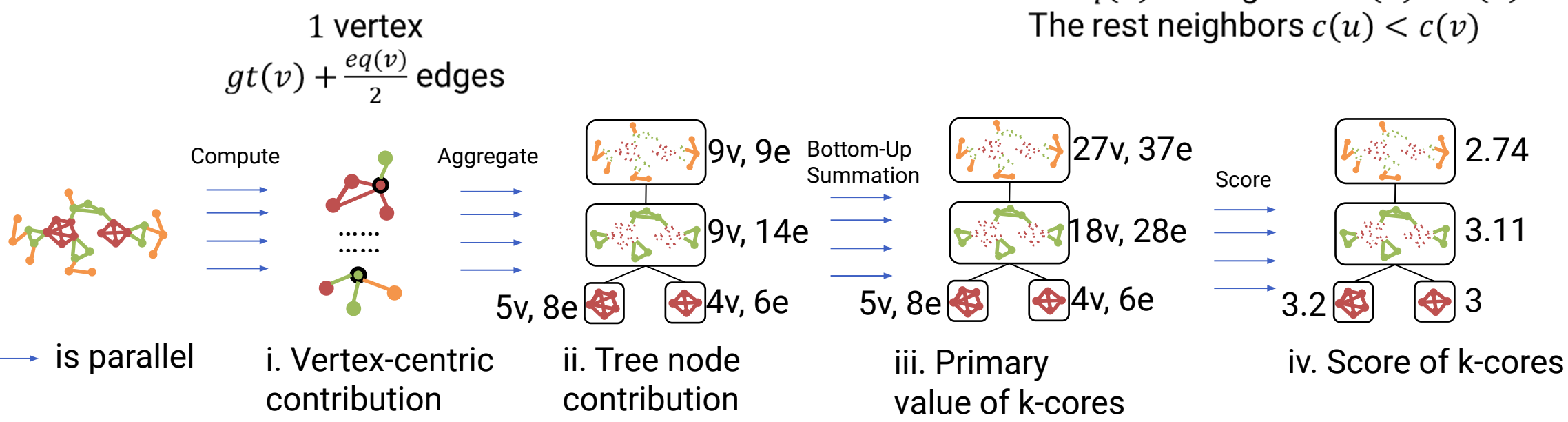
f: primary value, e.g., #edges, #vertices, #triangles…

3. Type-A Score Computation on HCD

E.g. Average Degree, #vertices, #edges

Preprocessing of every v (use in step i):
- $gt(v)$: # neighbors $c(u) > c(v)$
- $eq(v)$: # neighbors $c(u) = c(v)$

The rest neighbors $c(u) < c(v)$

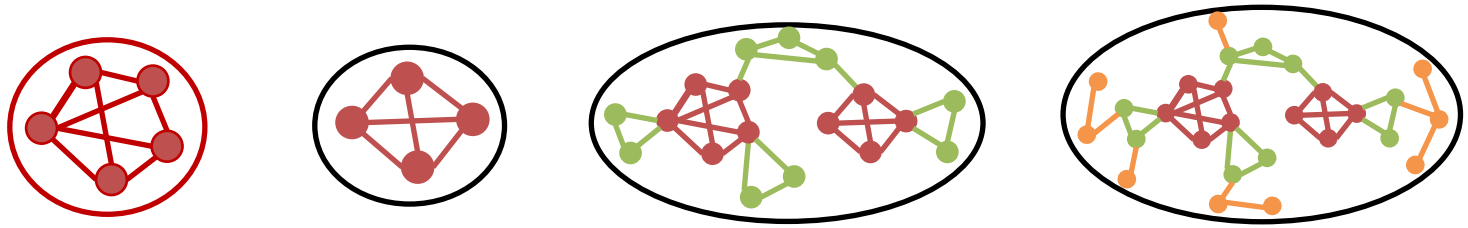


3. Type-B Score Computation on HCD

- Similar but more complex compared with type-A
- Triangle: use lower-degree endpoint to check triangle existence
- Triplet: count triplets based on the coreness of neighbors

Our Solution

1. Compute HCD
2. Preprocessing
3. Score Computation on HCD
4. Output the Best K-Core



The highest score

PBKS Time Analysis

- Preprocessing: $O(m)$ work
- Score Computation: Type-A: $O(n)$ work, Type-B: $O(m^{1.5})$ work
  - Work-efficient

$\boldsymbol{n}$: #vertices $\boldsymbol{m}$: #edges
Work: #steps
Work-Efficient: *#steps matches the best serial time complexity*

Roadmap

- Problem Definition
- Parallel Construction of HCD
- Parallel Subgraph Search on HCD
- ✔ Experimental Results
  - Experimental Setting
  - Runtime Performance
  - Application Performance
- Conclusion

Dataset Statistics & Experimental Setting

- 10 Public Networks, Up to **100M nodes & 4B edges**
- A Quad-Core (up to 40 threads) Linux server with 128G memory

Dataset	n	m	d_{avg}	k_{max}	$\|T\|$
As-Skitter	1,696,415	11,095,298	13.1	111	902
LiveJournal	3,997,962	34,681,189	17.3	360	1755
Hollywood	1,069,126	56,306,653	105.3	2208	678
Orkut	3,072,441	117,185,083	76.3	253	253
Human-Jung	784,262	267,844,669	683.0	1200	4087
Arabic-2005	22,744,080	639,999,458	56.3	3247	28693
IT-2004	41,291,594	1,150,725,436	55.7	3224	53023
FriendSter	65,608,366	1,806,067,135	55.1	304	450
SK-2005	50,636,154	1,949,412,601	77.0	4510	14356
UK-2007-05	105,896,555	3,738,733,648	70.6	5704	79318

Parallel HCD Construction Time

- Serial PHCD is 1.24-2.33x faster than `LCPS`
- 40-cores
  - Small graphs: 10x
  - Large graphs: 15-20x

Significantly faster than `LCPS`

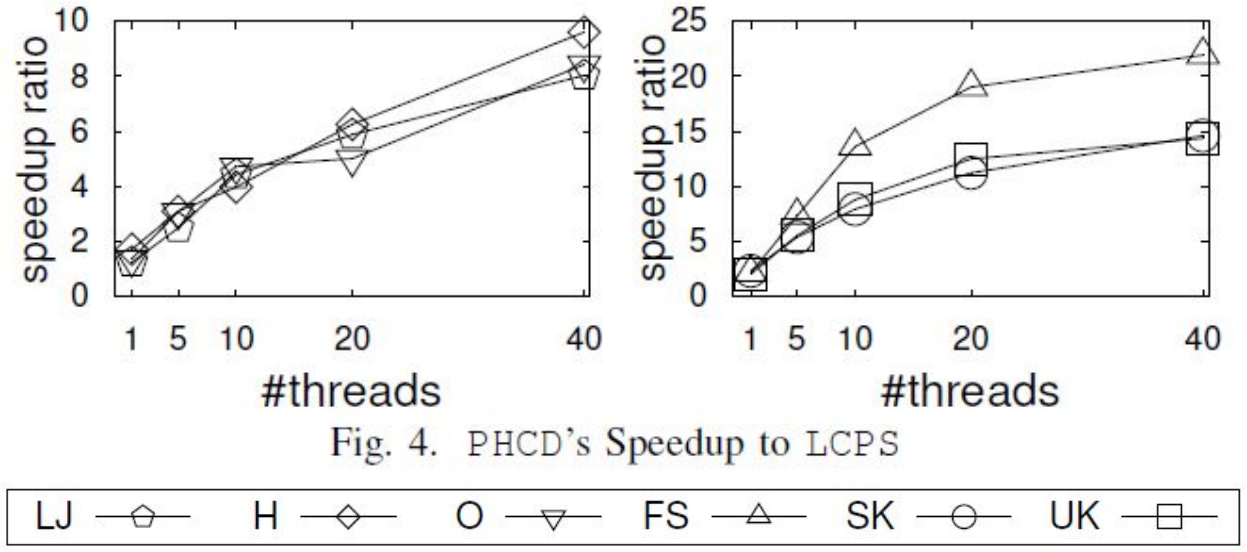


Fig. 4. PHCD's Speedup to LCPS

Parallel HCD Construction Time

- `LB`: the lower-bound cost of the UF-based method
  - `LB` is ~0.5 the cost of our method

Our `PHCD` is close to lower bound

TABLE III
TIME COST OF HCD CONSTRUCTION.

Dataset	(1)			(40)		
	PHCD (s)	LB	LCPS	PHCD (s)	LB	RC
AS	0.300	0.30x	1.66x	0.071	0.55x	4.06x
LJ	1.272	0.36x	1.24x	0.197	0.65x	9.11x
H	0.700	0.47x	1.71x	0.125	0.57x	63.82x
O	2.518	0.35x	1.37x	0.447	0.28x	25.35x
HJ	2.224	0.48x	2.05x	0.296	0.37x	124.97x
A	5.808	0.48x	1.87x	1.208	0.44x	9.46x
IT	10.885	0.44x	1.84x	1.766	0.54x	13.37x
FS	90.730	0.54x	2.12x	8.778	0.77x	58.74x
SK	16.372	0.50x	2.33x	2.609	0.63x	20.23x
UK	37.580	0.43x	2.02x	5.299	0.52x	22.43x

Parallel HCD Construction Time

- `LB`: the lower-bound cost of the UF-based method
  - `LB` is ~0.5 the cost of our method

Our `PHCD` is close to lower bound

- `RC` is required in Divide and Conquer method

D&C paradigm is inefficient in building HCD

TABLE III
TIME COST OF HCD CONSTRUCTION.

Dataset	(1)			(40)		
	PHCD (s)	LB	LCPS	PHCD (s)	LB	RC
AS	0.300	0.30x	1.66x	0.071	0.55x	4.06x
LJ	1.272	0.36x	1.24x	0.197	0.65x	9.11x
H	0.700	0.47x	1.71x	0.125	0.57x	63.82x
O	2.518	0.35x	1.37x	0.447	0.28x	25.35x
HJ	2.224	0.48x	2.05x	0.296	0.37x	124.97x
A	5.808	0.48x	1.87x	1.208	0.44x	9.46x
IT	10.885	0.44x	1.84x	1.766	0.54x	13.37x
FS	90.730	0.54x	2.12x	8.778	0.77x	58.74x
SK	16.372	0.50x	2.33x	2.609	0.63x	20.23x
UK	37.580	0.43x	2.02x	5.299	0.52x	22.43x

Parallel Subgraph Search Time

Score Computation (40-cores)

- Type-A: Small 30-45x
  Large 50x
- Type-B: ~20x

Significantly faster than BKS

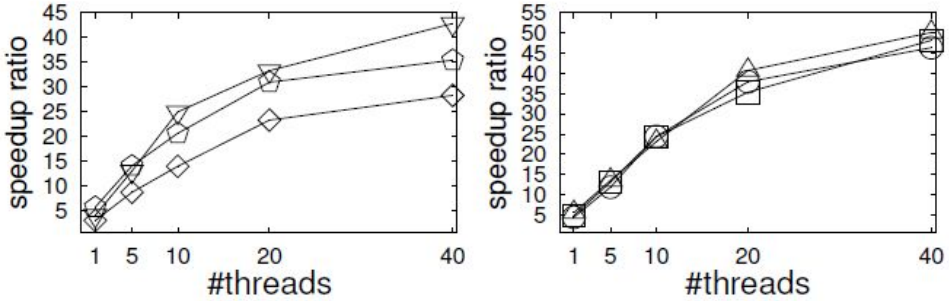


Fig. 6. PBKS's Speedup to BKS (Type-A)

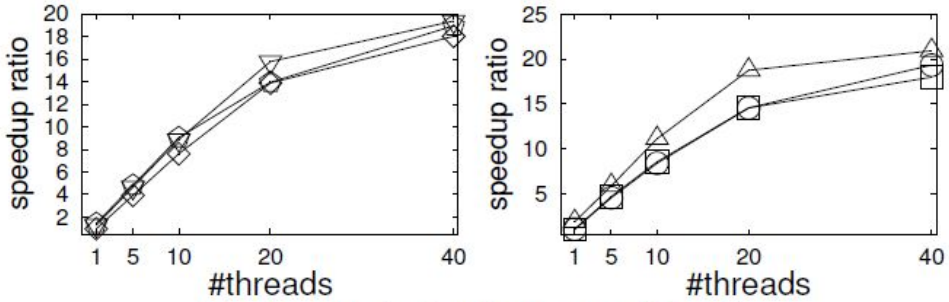


Fig. 8. PBKS's Speedup to BKS (Type-B)

Application: **D**ensest **S**ubgraph

DS: find the subgraph with the highest average degree

- **`CoreApp`**: a recent approximate solution [1] **`Opt-D`**: SOTA serial solution [2]
- **`PBKS-D`**: our subgraph search of the k-core with the highest average degree

Our Solution `PBKS-D`:

1. a $\frac{1}{2}$-approximate solution
2. outputs denser subgraph
3. 10x faster

Outperforms existing solutions

Dataset	CoreApp		Opt-D	PBKS-D	
	d_{avg}	time (s)	time (s)	d_{avg}	time (s)
AS	150.02	1.145	1.374	**178.801**	**0.196**
LJ	374.71	4.943	4.832	**387.027**	**0.529**
H	**2208**	3.002	3.635	**2208**	**0.542**
O	438.64	20.14	11.72	**455.732**	**1.159**
HJ	2013.88	15.272	14.457	**2114.915**	**2.851**
A	3247	40.703	35.359	**3248.92**	**4.511**
IT	3238.921	90.86	77.276	**4016.37**	**8.036**
FS	513.85	1041.528	836.279	**547.035**	**30.022**
SK	4513.00	202.682	125.04	**4514.99**	**12.890**
UK	5704	300.67	299.186	**5704.99**	**24.243**

d_{avg} **is average degree** **Better is bold**

`Opt-D`, `PBKS-D` have equal output

[1] Fang, Yixiang, et al. Efficient algorithms for densest subgraph discovery. PVLDB 2019.
[2] D. Chu, F. Zhang, X. Lin, W. Zhang, Y. Zhang, Y. Xia, and C. Zhang, "Finding the best k in core decomposition: A time and space optimal solution," in ICDE. IEEE, 2020, pp. 685–696.

Application: **M**aximum **C**lique

MC: find the largest subgraph where every pair of nodes is adjacent (NP-Hard)

- **PBKS-D**: our subgraph search of the k-core with the highest average degree
- S^* is the output of PBKS-D

The Output Subgraph S^* :
1. Contains MC in 7/10 datasets
2. Size of S^* is on average 1% of G

Potential pruning strategy for MC

Dataset	PBKS-D (output S^*)	
	$MC \subseteq S^*$	$\lvert S^* \rvert / n$
AS		0.027%
LJ	✓	0.011%
H	✓	0.207%
O		0.854%
HJ	✓	1.147%
A	✓	0.014%
IT	✓	0.010%
FS		0.08%
SK	✓	0.009%
UK	✓	0.005%

S^* contains the MC

The node proportion of S^* in the whole graph

Roadmap

- Problem Definition
- Parallel Construction of HCD
- Parallel Subgraph Search on HCD
- Experimental Results
- ✔ Conclusion

Conclusion

☑ Parallel HCD Construction

- ***P-Complete Proof (difficult to parallelize effectively)***
- ***First parallel algorithm:*** near-linear work $O(n\sqrt{p} + m\alpha(n) + F)$

☑ Parallel Subgraph Search on HCD

- ***First parallel algorithm***
- ***Score Computation:*** work-efficient

  $O(n)$ for type-A, $O(m^{1.5})$ for type-B, after $O(m)$ preprocessing

☑ Useful in Related Problems

- ***Application***: Approximate Densest Subgraph, Maximum Clique
- ***Runtime:*** Outperform existing works significantly

Q & A