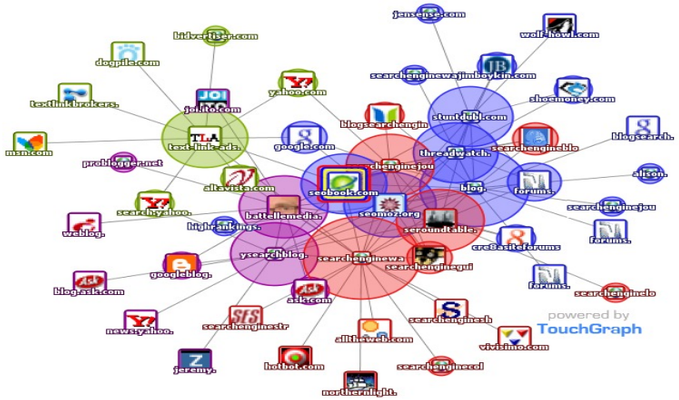
Parallel Colorful h-star Core Maintenance in Dynamic Graphs

Sen Gao, Hongchao Qin,
Rong-Hua Li, Bingsheng He

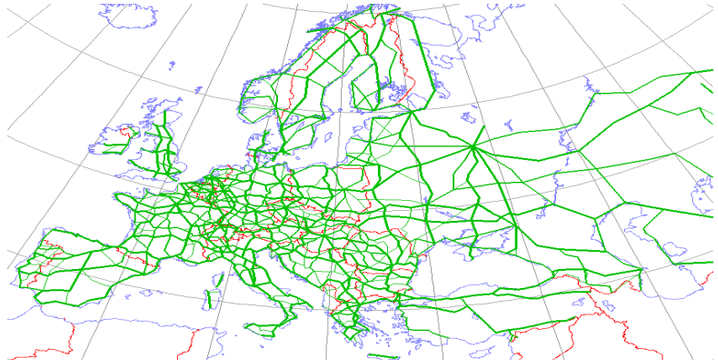
Presenter: Sen Gao

Graph data are everywhere!

Social Network

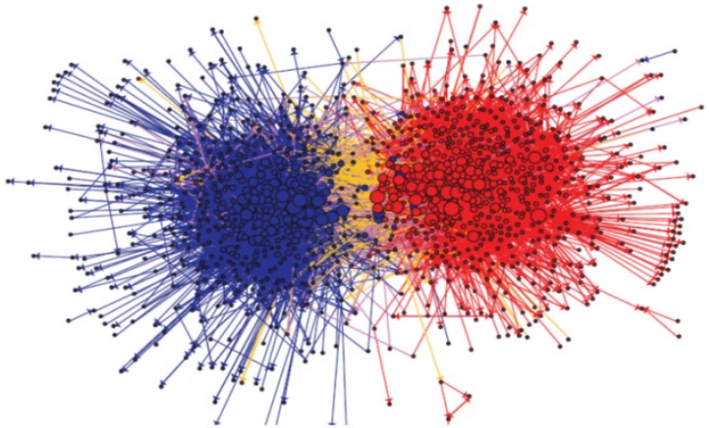


Internet

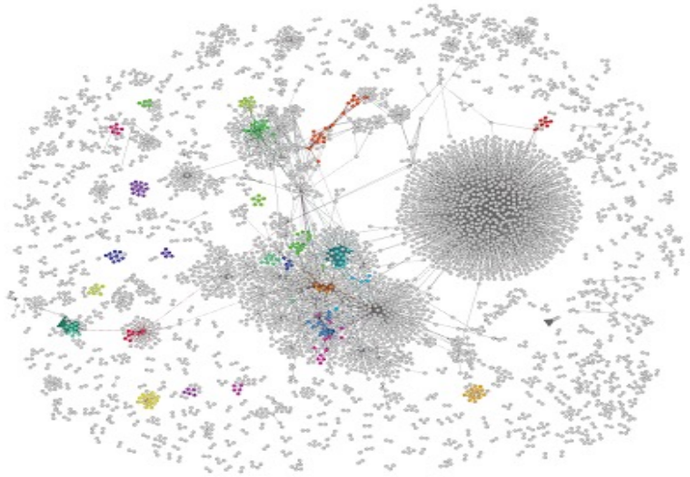


Road Network

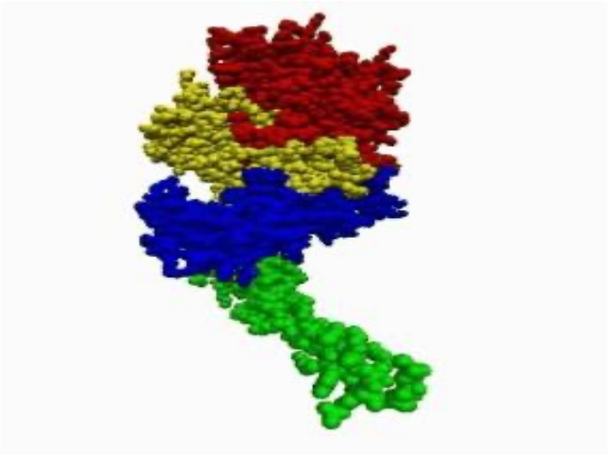
IoT Network



Blog Network



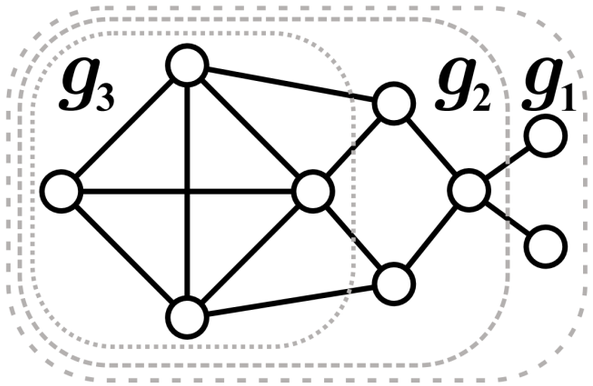
Twitter Social Network



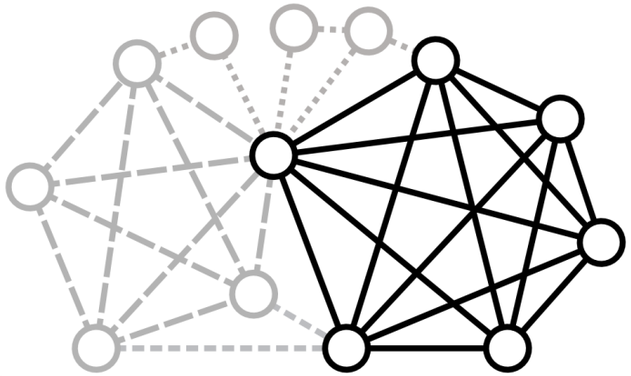
PPI Network

Real-life networks often contains cohesive subgraph structures

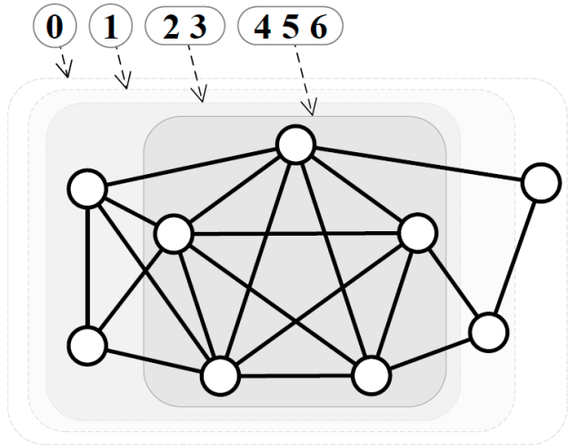
Existing cohesive subgraph models



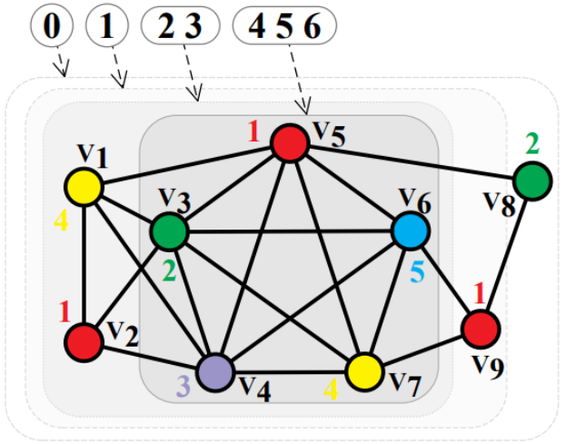
k-core



k-truss



h-clique *k* core (*h*=3)



colorful *h*-star *k* core (*h*=3)

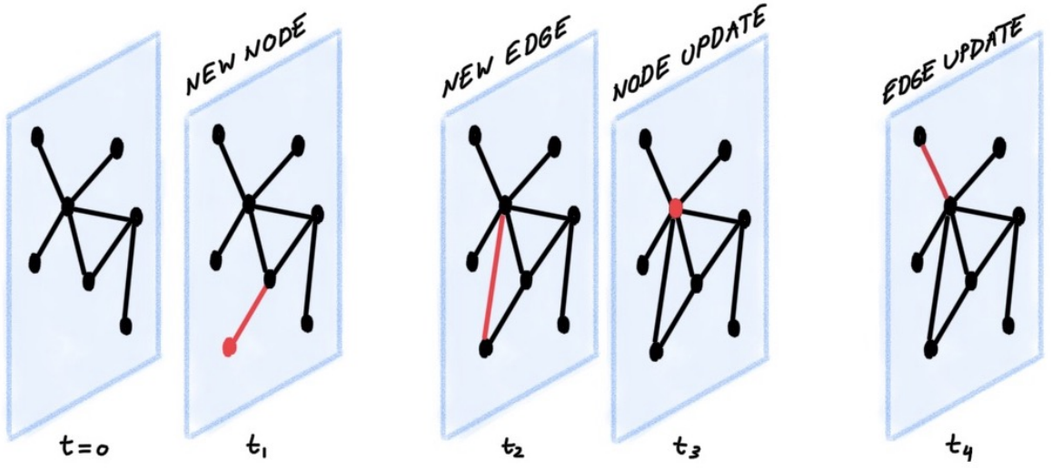
Comparison between cohesive subgraph models

Models	Minimum Unit	Analysis Type	🙂	😕
***k*-core**	Node	*Lower*-order	**Low complexity**	**Cannot locate complex regions**
***k*-truss**	Edge	*Lower*-order		
***h*-clique *k* core**	*h*-clique	*Higher*-order	**Higher-order analysis**	**Costly computation**
Colorful *h*-star *k* core	colorful *h*-star	*Higher*-order	**Higher-order analysis Efficient**	——

Existing algorithms compute colorful *h*-star cores by *Peeling*

Limitations

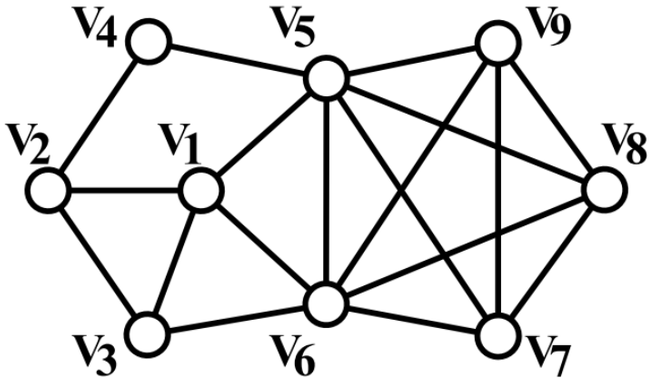
- **Low scalability (inherently sequential)**
- **Low efficiency for dynamic graphs (compute from scratch)**



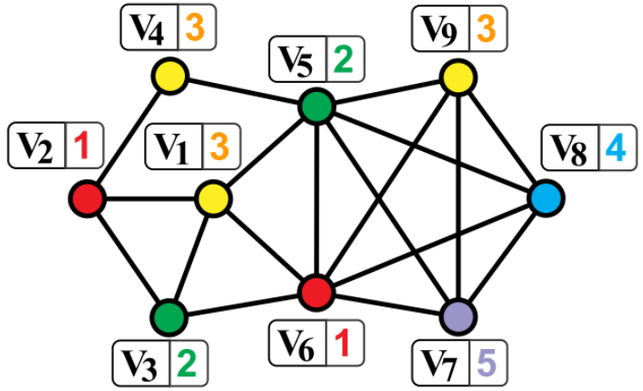
Our contributions

New Concept	Colorful h-star n-order H-index, with a thorough theoretical analysis
Novel Algorithms	• H-index based Core decomposition Parallel local algorithm Three optimizations • H-index based Core maintenance Tight lower and upper bounds for edge update
Extensive experiments	• Experiments on 14 large datasets, three orders of magnitude speedup • Case studies on 4 real-world graphs

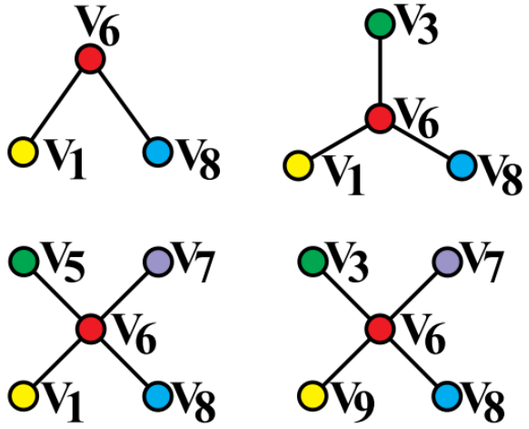
A colored graph and its colorful 3-star cores



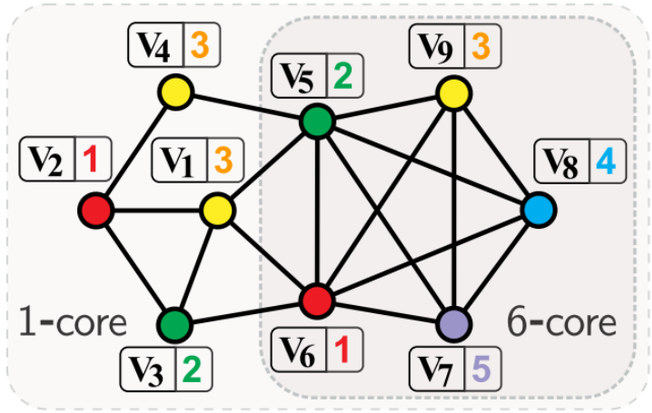
(a) An undirected graph



(b) A colored graph



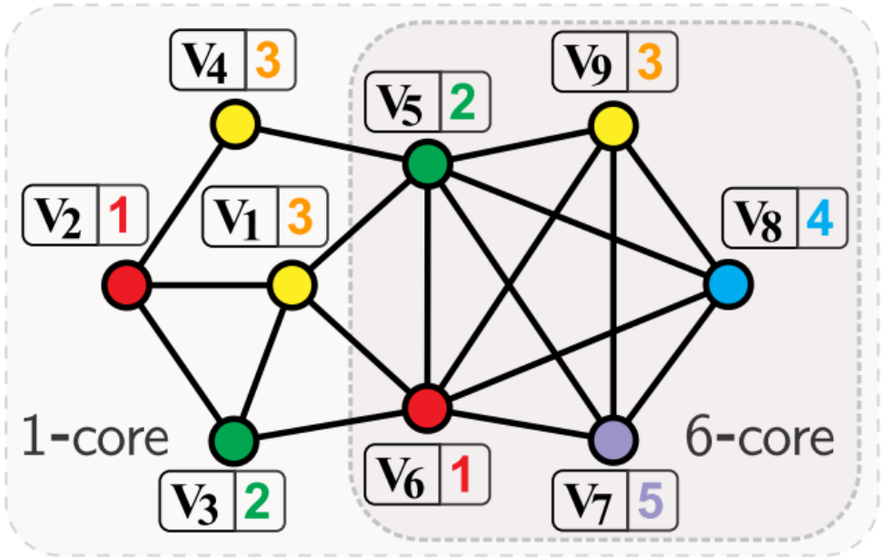
(c) Colorful stars of v_6



(d) Colorful 3-star cores

Observation

The core number of u (c_u) can be computed *only from* the core numbers of its neighbors



This computation has two cases:

1) count colorful h-stars only on the neighbors with larger core numbers than u. (c_{v_1} *can be derived from* v_5 *and* v_6)

2) Set it to the core number of one of u 's neighbors. ($c_{v_4} = c_{v_2}$)

Colorful h-star n-order H-index

For a node u, define $H_u^{(n)}(G,h)$ following recurrence relation.

$$\mathrm{H}_u^{(n)}(G,h) = \begin{cases} d_u(G,\mathcal{S}) & n = 0 \\ \min(\mathrm{H}_{w^{(n-1)}}^{(n-1)}(G,h), \mathrm{DP}^{(n-1)}(p^{(n-1)})) & n > 0 \end{cases}$$

Case 2 Case 1

Here $P^{(n)}$ is a neighbor index of the sorted adjacency list.

$$p^{(n)} = \min\{i \in \{1,2,\cdots,d_u(G)\} : !(\mathrm{DP}^{(n)}(i) < \mathrm{H}_{v_i^{(n)}}^{(n)}(G,h) \wedge \mathrm{DP}^{(n)}(i) < \mathrm{H}_{v_{i+1}^{(n)}}^{(n)}(G,h))\},$$

Colorful h-star n-order H-index

Theoretical Analysis

- **MONOTONICITY**

$$\boldsymbol{H}_u^{(n)}(G,h) \leq \boldsymbol{H}_u^{(n-1)}(G,h)$$

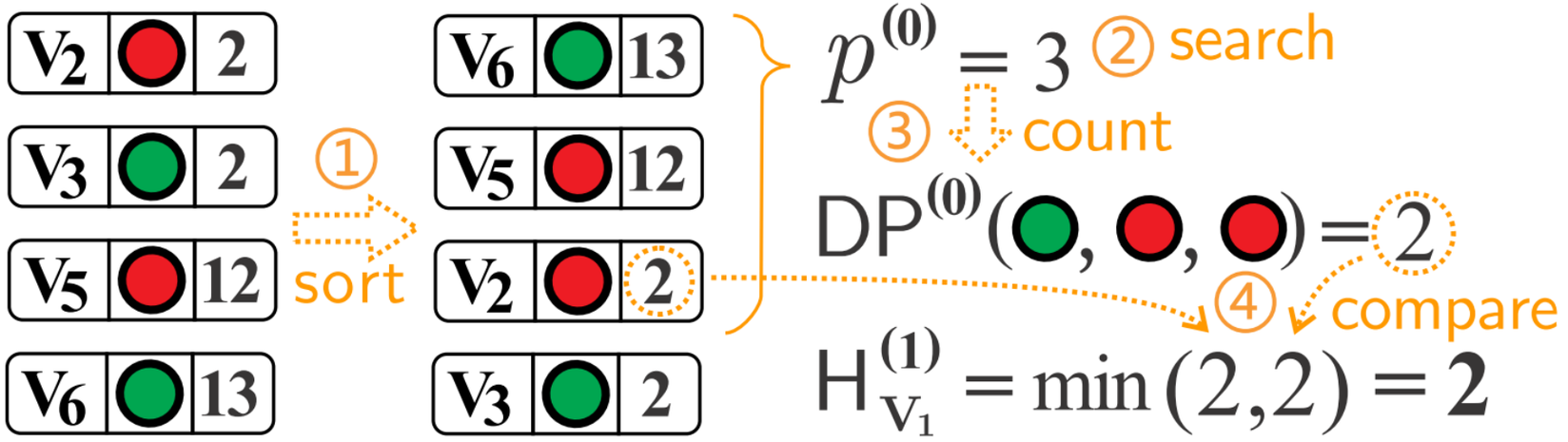
- **CONVERGENCE**

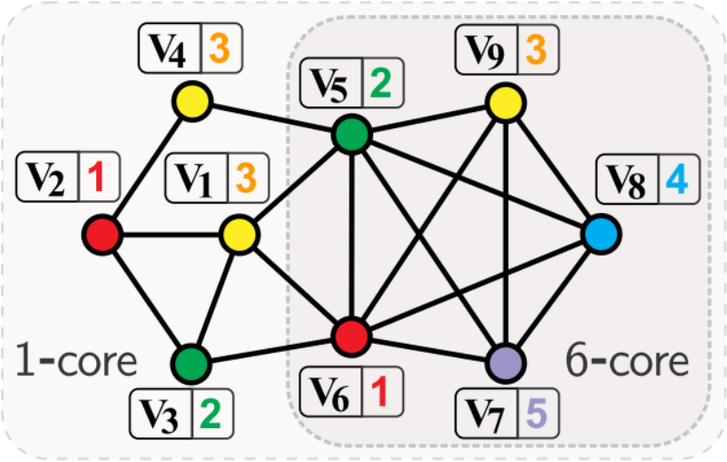
$$\lim_{n\to\infty} \boldsymbol{H}_u^{(n)}(G,h) = \boldsymbol{c_u}(\boldsymbol{G},\boldsymbol{S})$$

- **THEORETICAL CONVERGENCE BOUND**

Parallel Algorithm for Colorful h-star Core Decomposition

Example





Neighbors' 0-order H-index → v_1's 1-order H-index

Three Optimizations

Optimization	Type	Design Goal
Asynchronous computing	Inter-iteration	Reduce the number of iterations
Processing ordering heuristic		
Pruning technique	Intra-iteration	Avoid redundant computations within an iteration

Three Optimizations

1. **Asynchronous computing**

   In the *i*-th iteration, H-index computation consider neighbors' H-index in both the (*i*-1)-th and the *i*-th iteration.

2. **Processing ordering heuristic**

   For nodes of large degrees, more neighbors are processed in the *i*-th iteration, smaller H-indexes they will get.

3. **Pruning technique**

   Compute H-index based on part of neighbors

Parallel Algorithm and Optimizations Running on the Example Graph ($h = 3$, ☑: computation pruned)

Methods	OPT Types	$H_v^{(n)}(G,3)$	n-order H-index									#Iterations	#invocations
			v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9		
		$H^{(0)}$	4	2	2	1	12	13	6	6	6		
Local	None	$H^{(1)}$	2	1	2	1	6	6	6	6	6	4	36
		$H^{(2)}$	2	1	1	1	6	6	6	6	6		
		$H^{(3)}$	1	1	1	1	6	6	6	6	6		
		$H^{(4)}$	1	1	1	1	6	6	6	6	6		
OPT-1	*inter*	$H^{(1)}$	2	1	1	1	6	6	6	6	6	3	27
		$H^{(2)}$	1	1	1	1	6	6	6	6	6		
		$H^{(3)}$	1	1	1	1	6	6	6	6	6		
OPT-2		$H^{(1)}$	1	1	1	1	6	6	6	6	6	2	18
		$H^{(2)}$	1	1	1	1	6	6	6	6	6		
OPT-3	*intra*	$H^{(1)}$	2	1	2	1	6	6	6	6	6	4	11
		$H^{(2)}$	☑	☑	1	☑	☑	☑	☑	☑	☑		
		$H^{(3)}$	1	☑	☑	☑	☑	☑	☑	☑	☑		
		$H^{(4)}$	☑	☑	☑	☑	☑	☑	☑	☑	☑		
Local + OPT-1-2-3	*inter intra*	$H^{(1)}$	1	1	1	1	6	6	6	6	6	2	9
		$H^{(2)}$	☑	☑	☑	☑	☑	☑	☑	☑	☑		

Challenges of colorful *h*-star core maintenance problem

For each edge update (Deletion/Insertion)

- Recomputing core numbers of all nodes from scratch is **costly.**
- Identifying unaffected nodes is quite **hard.**
- The changes of core numbers may be much **larger than 1.**

Solution - *two-stage* updating algorithms

1. Identify affected nodes based on **our proposed theorems.**
2. Accelerate the convergence of updating these nodes by **designing lower and upper bounds** based on original core numbers.

Experiments

Datasets (1K=10^3, 1M=10^6, 1B=10^9)

Datasets	$n = \|V\|$	$m = \|E\|$	χ	d_{max}	d_{avg}	Description
Buzznet	101.2K	2.8M	62	64.3K	55	Online social activities
Flickr	514K	3.2M	107	4.4K	12	
Digg	770.8K	5.9M	66	17.6K	15	
Orkut	3M	106.3M	79	27.5K	71	
LiveJournal	4.8M	42.9M	324	20.3K	18	
Twitter	41.6M	1.2B	1084	3.0M	57.7	
Nasasrb	54.9K	1.3M	38	275	48	Scientific computing
Pkustk	87.8K	2.6M	54	131	58	
Pwtk	217.9K	5.7M	42	179	52	
MsDoor	404.8K	9.4M	42	76	46	
LDoor	909.5K	20.8M	42	76	46	
DBLP	317.1K	1M	114	343	7	Collaboration
Skitter	1.7M	11.1M	71	35.5K	13	Internet topology
Patent	3.8M	16.5M	14	793	9	Citation

- 14 large real-world graphs with billions of edges
- Various domains

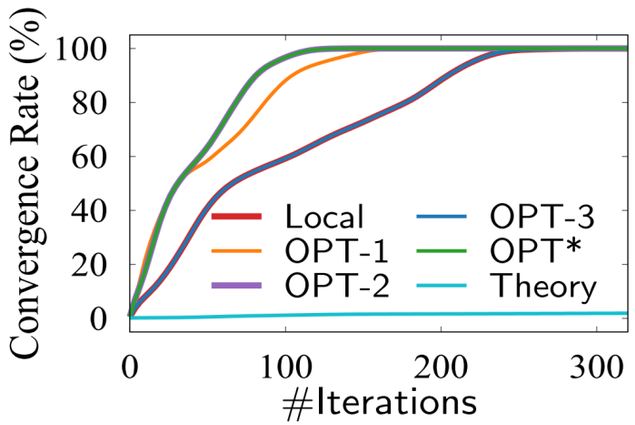
Convergence Evaluation (Overview)

Our best method OPT∗ takes the least iterations and time.

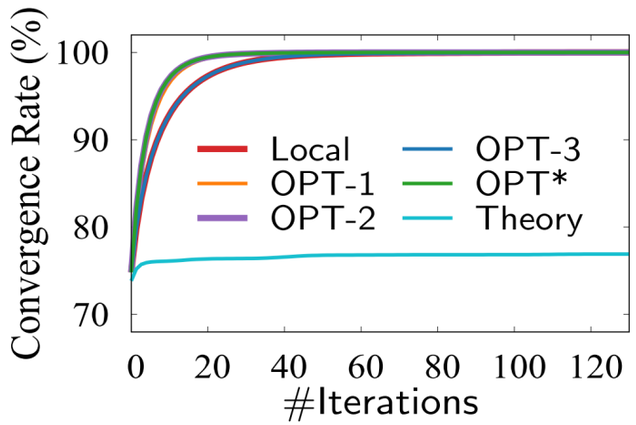
Datasets	#Iterations						Average Invocations					Time (sec)				
	Theory	Local	OPT-1	OPT-2	OPT-3	OPT∗	Local	OPT-1	OPT-2	OPT-3	OPT∗	Local	OPT-1	OPT-2	OPT-3	OPT∗
Buzznet @	30234	66	35	33	66	33	33.76	17.90	16.88	1.81	1.19	16	8.6	7.9	4.5	2.4
Flickr	24686	85	45	45	85	45	8.20	4.34	4.34	0.59	0.35	19.6	10.6	10	4.9	2.6
Digg	39341	83	44	40	83	40	8.21	4.35	3.96	0.55	0.34	36.7	20.2	17.6	11.7	6.2
Orkut	1245333	237	117	119	237	119	197.53	97.52	99.19	9.58	5.33	2860.5	1422.1	1456.8	425.4	227
LiveJournal	397783	122	65	62	122	62	29.21	15.57	14.86	1.72	1.02	455	244.2	229.1	72.5	38.9
Twitter	4768506	141	73	71	141	71	56.21	29.1	28.31	0.89	0.68	31540.1	15280.1	14170.2	2477.5	1274.7
Nasasrb	16797	285	170	127	285	127	284.52	169.72	126.79	26.94	17.49	32.8	19.5	15.2	4.3	3
Pkustk	5032	91	43	39	91	39	91.00	43.00	39.00	7.15	5.48	20.5	10	9.1	2.6	2
Pwtk	15421	289	74	69	289	69	288.39	73.84	68.86	28.94	14.50	144.3	36	34.5	19.1	8.9
MsDoor	27107	166	88	93	166	93	166.00	88.00	93.00	6.14	4.67	137.9	74.1	79.1	8.5	6.5
LDoor	77066	235	120	122	235	122	235.00	120.00	122.00	7.19	5.54	434.9	224.7	231.9	23.5	18.4
DBLP	880	30	17	19	30	19	1.89	1.07	1.20	0.11	0.09	1	0.6	0.6	0.2	0.2
Skitter	46157	79	43	43	79	43	6.36	3.46	3.46	0.30	0.21	33.6	18.9	18.5	6.8	4.1
Patent	1492	69	37	37	69	37	0.06	0.04	0.04	0.02	0.02	3.6	3.2	2.2	3.2	2.1

Convergence Rate over Iterations

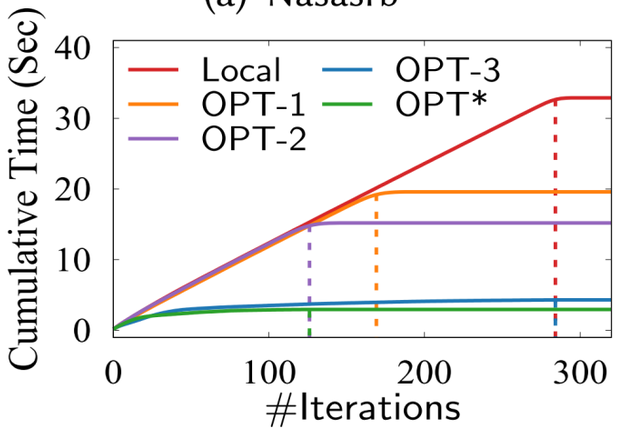
Three optimizations works independently for accelerating convergence.



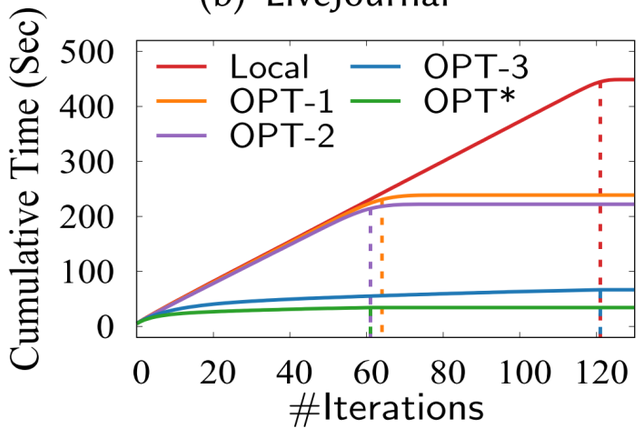
(a) Nasasrb



(b) LiveJournal



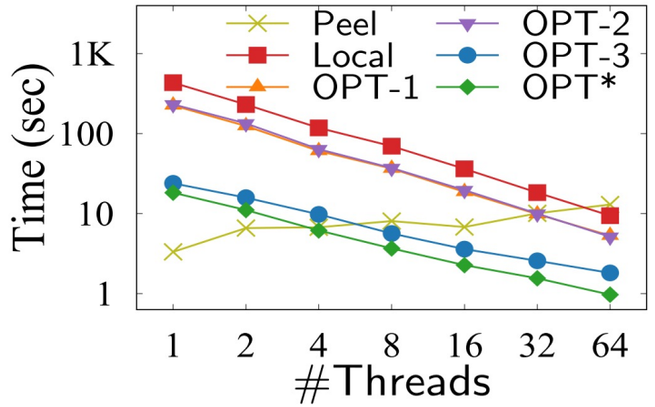
(c) Nasasrb



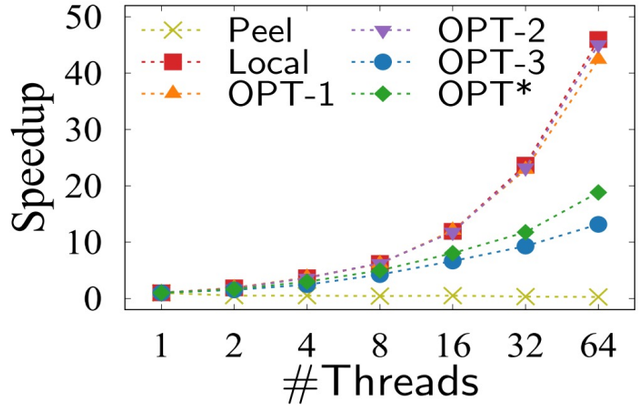
(d) LiveJournal

Parallelization Performance Evaluation

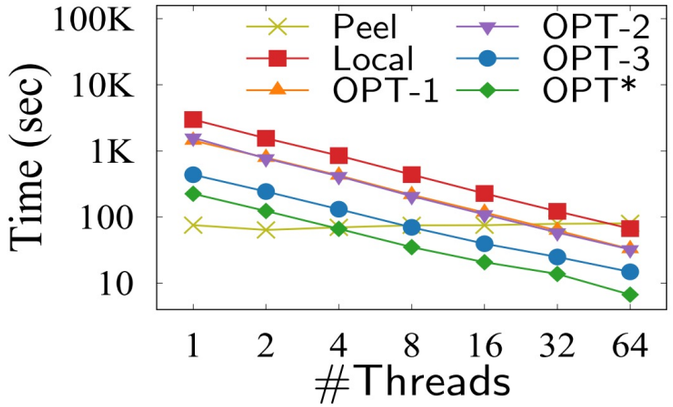
Our algorithms all achieve a significantly high degree of parallelism.



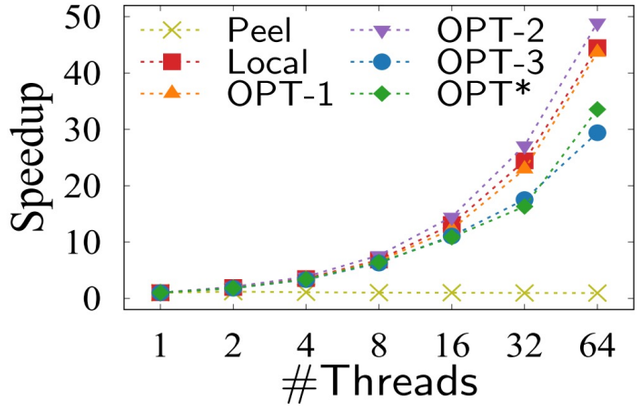
(a) LDoor



(b) LDoor



(c) Orkut

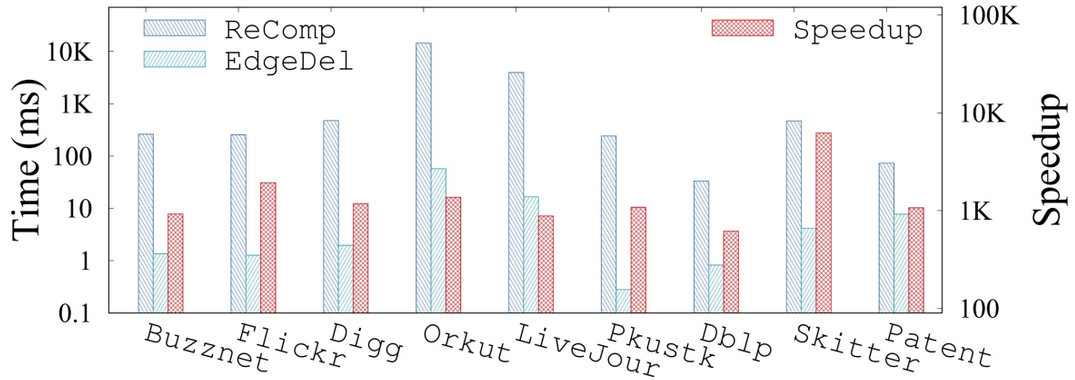


(d) Orkut

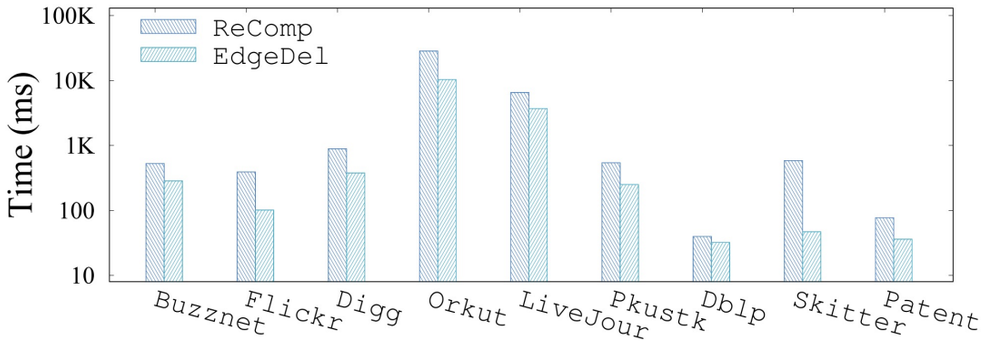
Maintenance Evaluation - Random Edge Update

Both edge deletion and insertion algorithms achieve three orders of magnitude speedup.

Edge Deletion

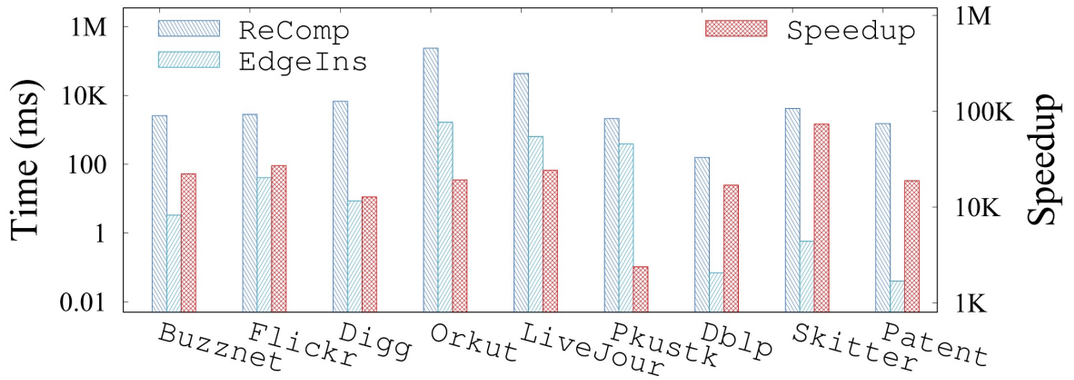


(a) Edge Deletion (Single)

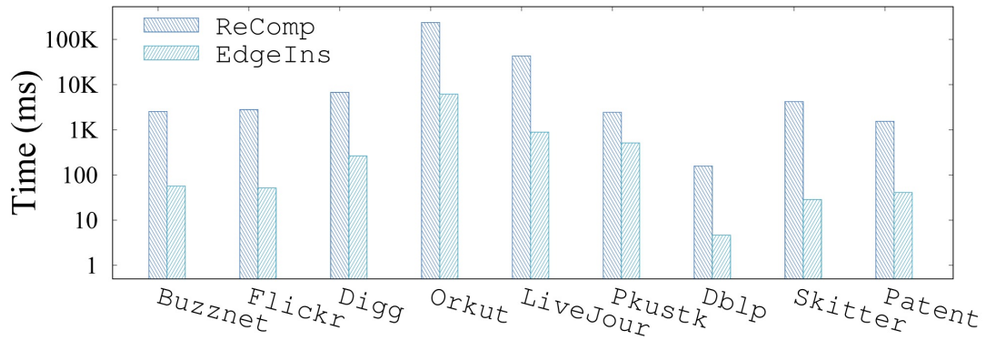


(b) Edge Deletion (Batch)

Edge Insertion



(a) Edge Insertion (Single)



(b) Edge Insertion (Batch)

Maintenance Evaluation - Skewed Edge Update

Skewed Updates: The edge deletion/insertion that affects the most nodes.

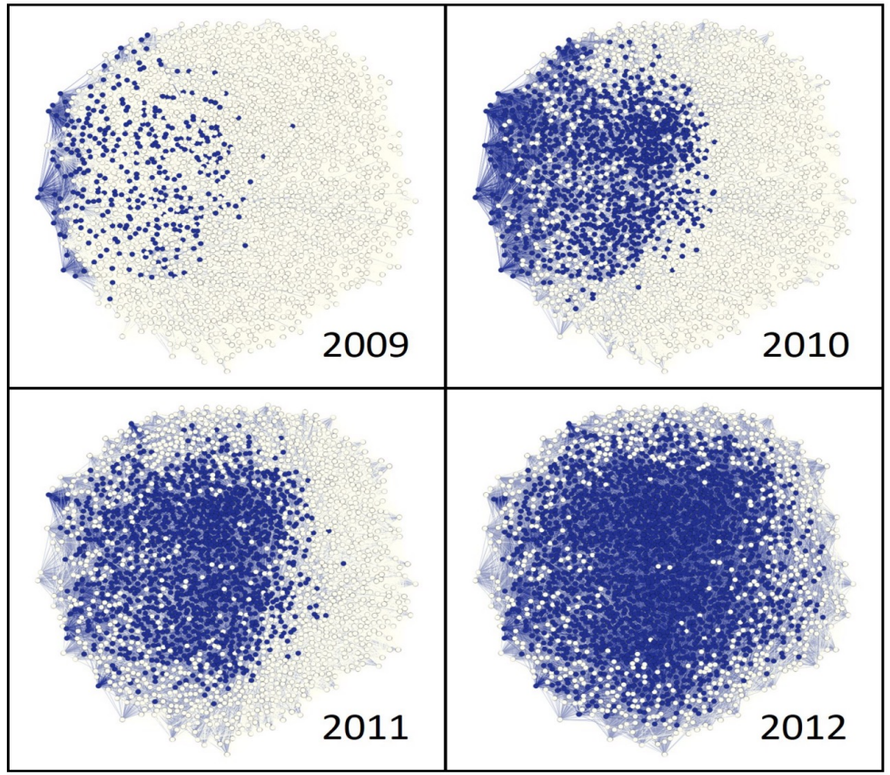
Updating time of skewed updates on graphs with skewed structures

Dataset	Type	Delete					Insert				
		ReComp	Random		Skew		ReComp	Random		Skew	
			%	EdgeDel	%	EdgeDel		%	EdgeIns	%	EdgeIns
Skitter	Power law	183.2	*0.1*	0.11	*1.2*	11.6	686.9	*1.9*	12	*94.0*	559.3
Digg		101.5	*1.8e-5*	0.12	*0.5*	1.7	447.8	*0.4*	11.6	*39.2*	70.1
Twitter		44K	*2.4e-7*	14.7	*1.9e-6*	265.3	107K	*1.2e-5*	21.5	*31.5*	11K
Pwtk	Skew	122.2	*1.2*	0.8	*6.2*	3.8	806.6	*84.4*	665.7	*87.9*	837.2
MsDoor		267.2	*0.7*	1.1	*6.4*	10.1	508.5	*35.8*	182.2	*58.9*	243.6

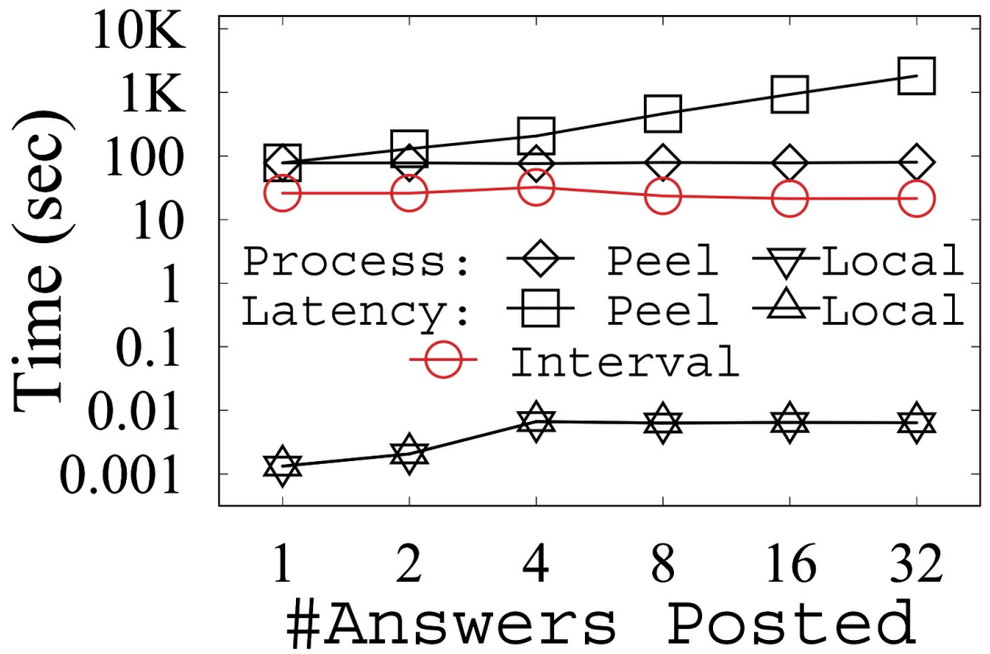
- All methods take more time for skewed updates.
- Our algorithms still outperform the baseline.

Case Study on Stack Overflow

Revelation of active users in social networks and real-time capture of graph changes



(a) Active user distribution in four years



(b) Average time for temporal edge insertions

Conclusion

- We study the parallel colorful h-star core decomposition and maintenance problem.
- We design a new concept, based on which we develop efficient parallel algorithms and optimizations. Furthermore, they are extended for graph updates.
- Experimental results show our methods are significantly faster and more scalable than baselines.

Thanks