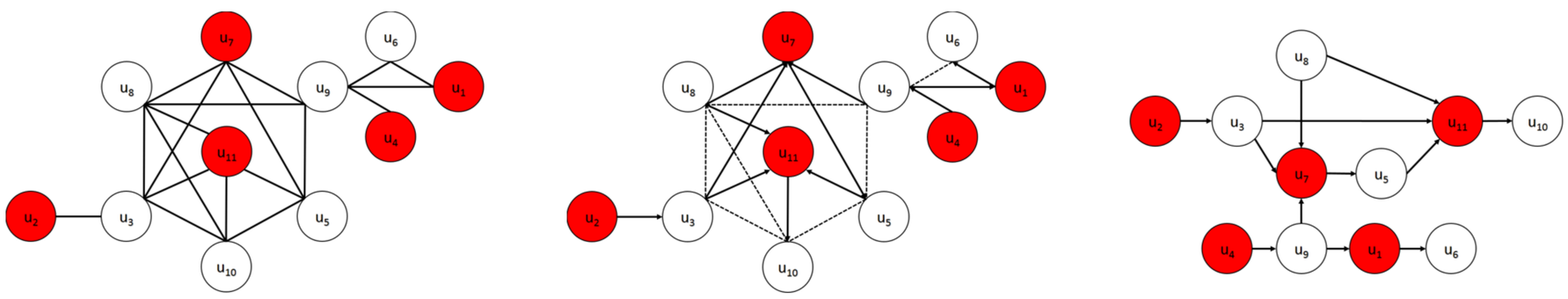
353: Computing A Near-Maximum Independent Set In Dynamic Graphs

Weiguo Zheng[1,2] **Chengzhi Piao**[1] **Hong Cheng**[1] **Jeffrey Xu Yu**[1]

[1]The Chinese University of Hong Kong, HK SAR, China
[2]Fudan University, China



1 Background

Independent Set

Given a set of vertices in graph G, we call it an independent set of G if and only if there are no neighborhoods among them.

Maximal independent set

Can not add more vertices into it.

Maximum independent set(MIS)

A MIS has the largest size among all independent sets.
Maybe not unique.

Problem Definition

Target: maintain a Near-MIS in dynamic graphs

Input

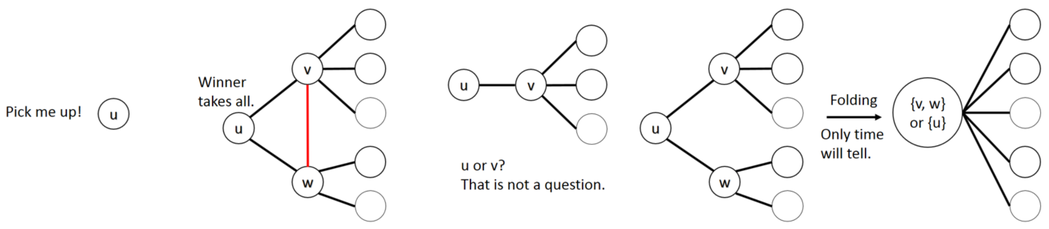
An undirected graph G_0 and the initial $NearMIS_0$
A sequence of updates on the graph structure: $update_i$

Output

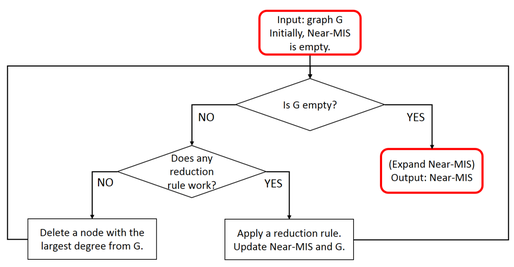
online maintain the Near-MIS after each update: $NearMIS_i$

2 Dependency Graph based Framework

Reduction Rule

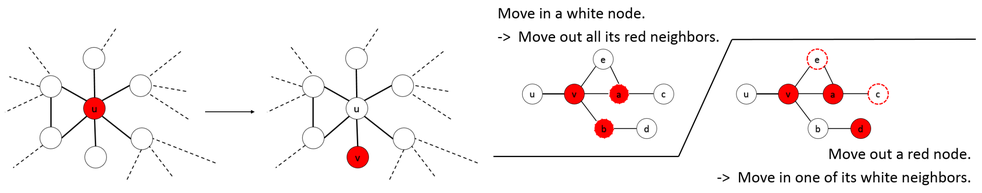


Reducing-Peeling Framework

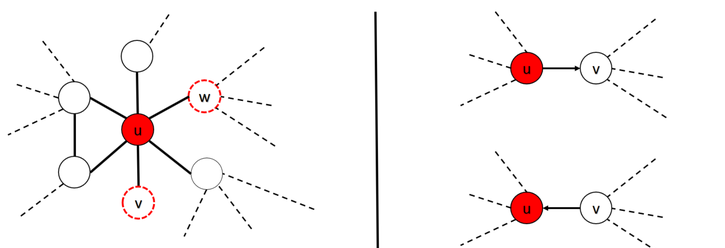


Brute Force Search: Move Out a Node from the MIS

To maintain a MIS, we must choose a neighbor to replace u.



Two Ways of Pruning



Definition of Dependency Graph

We call the nodes in the independent set reducing nodes.
Others are called dependent nodes.

1. Contain all edges between reducing and dependent nodes.
2. Each edge is directed.
3. Each reducing node has at most two out-neighbors.
4. Each dependent node has at most one in-neighbor.

DG: Construction

The reducing-peeling framework does not merely output a Near-MIS, but also implies a topological order of the graph.

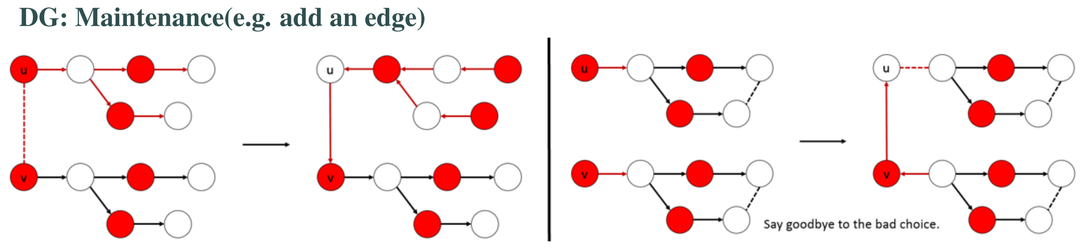
1. For each reducing node, its out-neighbors are decided when a reduction rule works.
2. Then, the others become its in-neighbors.

DG: Dealing with Updates

We mainly focus on the following three operations.

1. Delete a node u: If u is a reducing node, try to move out u.
2. Add an edge (u, v): If both u and v are reducing nodes, try to move out u or v. Simply abandon either if it fails.
3. Delete an edge (u, v): Try to move in the dependent node.

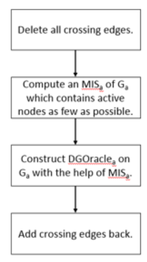
DG: Maintenance(e.g. add an edge)



Batch Update

G_a: affected subgraph
G_r: the remaining graph
Crossing Edge: an edge between G_a and G_r

For each crossing edge (u, v)
- u is in G_a and v is in G_r.
- v is a reducing node(v is in Near-MIS).
- Then u is called an active node.



Delete a node **not in Near-MIS$_0$** with the largest degree from G.

DGOracle

DGOracle

Use $NearMIS_0$ to guide the inexact reduction.

3 Results

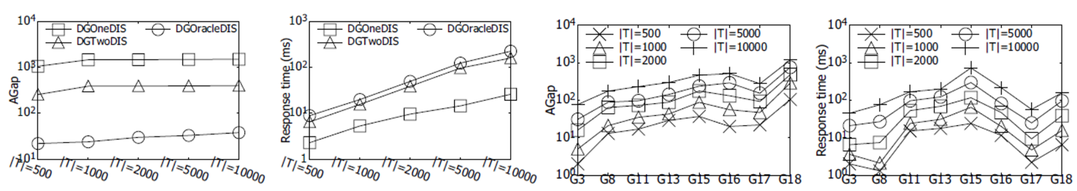
ID	Graph	|V(G)|	|E(G)|	d
G01	Wiki-Vote	7,115	100,762	28.32
G02	CA-HepPh	12,006	118,489	19.74
G03	AstroPh	18,771	198,050	21.1
G04	Brightkite	58,228	214,078	7.35
G05	Epinions	75,879	405,740	10.69
G06	Slashdot	82,168	504,230	12.27
G07	com-dblp	317,080	1,049,866	6.62
G08	com-amazon	334,863	925,872	5.53
G09	BerkStan	685,230	6,649,470	19.41
G10	web-Google	875,713	4,322,051	9.87
G11	soc-pokec	1,632,803	22,301,964	27.32
G12	as-skitter	1,696,415	11,095,298	13.08
G13	wiki-tomcats	1,791,489	25,444,207	28.41
G14	Wiki-Talk	2,394,385	4,659,565	3.89
G15	com-orkut	3,072,441	117,185,083	76.28
G16	cit-Patents	3,774,768	16,518,947	8.75
G17	com-lj	3,997,962	34,681,189	17.35
G18	LiveJournal	4,846,609	42,851,237	17.68

TABLE II
THE GAPS CAUSED BY 1000 UPDATES AND THE MEMORY USAGE

Graphs	*LSTwo+LazySearch*+		*DGOneDIS*		*DGTwoDIS*		*DGOracleDIS*		*BatchUE*		*BatchUN*		Memory Usage (MB)		
	AGap	*RGap*	*AGap*	*RGap*	*AGap*	*RGap*	*AGap*	*RGap*	*AGap*	*RGap*	*AGap*	*RGap*	*DGOneDIS*	*DGTwoDIS*	*BatchUE*
Wiki-Vote	23	23	1	0	1	1	1	1	1	0	1	1	2.48	2.51	2.52
CA-HepPh	62	62	10	9	6	6	6	6	5	2	4	-5	2.99	3.04	3.76
AstroPh	35	35	12	11	5	5	3	3	7	7	5	5	4.98	5.05	5.14
Brightkite	17	17	2	1	1	1	1	1	1	0	1	1	6.29	6.51	6.76
Epinions	19	19	1	1	0	0	0	0	1	0	1	0	11.09	11.39	11.48
Slashdot	26	26	3	1	2	0	0	0	1	1	2	1	13.49	13.81	14.04
com-dblp	72	72	22	10	6	5	5	5	5	4	6	4	31.59	32.81	32.93
com-amazon	89	89	150	-2	21	-2	0	0	3	-5	4	-5	29.18	30.45	30.86
BerkStan	838	838	1074	5	761	4	18	18	15	0	12	3	168.53	171.15	172.28
web-Google	306	306	314	18	173	7	10	10	11	5	16	5	119.8	123.14	124.21
as-skitter	371	371	393	-2	295	0	0	0	7	0	14	-1	294.39	300.86	302.38
Wiki-Talk	74	74	1	0	1	0	0	0	1	0	1	1	163.74	172.87	173.52
LiveJournal	414	414	1458	4	388	-2	2	2	10	-2	18	-2	1096.34	1114.83	1115.12

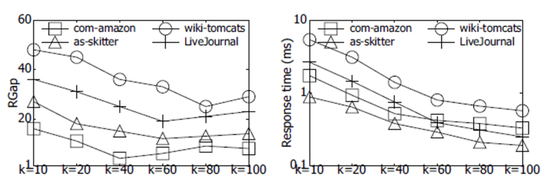
TABLE III
VISITED VERTICES ON AVERAGE AND RESPONSE TIME CONSUMED BY 1000 UPDATES

Graphs	*LSTwo+LazySearch*+	*DGOneDIS*		*DGTwoDIS*		*DGOracleDIS*		*BatchUE*		*BatchUN*		*DGOneDY*		*DGTwoDY*		*DGOracleDY*	
	ms	#	*ms*	#	*ms*	#	*ms*	#	*ms*	#	*ms*	#	*ms*	#	*ms*	#	*ms*
Wiki-Vote	19.3	1.1	1.3	5.6	7.2	3.6	4.5	1.3	1.5	0.28	0.3	1.3	1.5	5.5	7.1	3.9	4.6
CA-HepPh	36.6	2.7	3.2	3.7	4.6	4.4	5.4	1.7	1.9	0.78	0.9	2.6	3.3	4.2	4.8	4.9	5.7
AstroPh	63.7	1.4	1.5	2.8	3.3	5.8	6.9	0.76	0.9	0.31	0.4	1.3	1.5	3.1	3.7	6.7	7.1
Brightkite	104.6	3.6	4.1	3.9	4.5	7.3	8.3	1.8	2.2	0.91	1.1	4.0	4.6	3.8	4.5	6.9	7.7
Epinions	72.1	0.92	1.2	1.4	1.7	5.2	6.6	0.71	0.85	0.28	0.33	1.1	1.4	1.2	1.8	5.1	6.3
Slashdot	166.8	1.8	2.3	2.3	2.8	7.5	9.3	0.79	0.9	0.17	0.2	1.9	2.1	2.2	2.7	7.4	8.4
com-dblp	88.7	1.1	1.4	1.7	1.9	3.2	4.2	0.56	0.66	0.29	0.41	1.3	1.6	2.1	2.5	3.1	4.2
com-amazon	1209.3	2.1	2.6	2.2	2.7	4.3	5.1	1.2	1.4	0.41	0.52	2.2	2.8	2.3	2.9	4.3	5.6
BerkStan	2723.4	6.4	7.7	7.5	8.6	8.9	10.5	2.8	3.3	1.3	1.5	5.8	7.2	6.1	7.8	8.2	9.6
web-Google	4045.8	4.7	5.4	5.6	6.3	7.7	9.3	2.2	2.6	0.64	0.72	4.3	5.5	5.2	6.5	8.6	9.7
as-skitter	3659.3	7.2	8.6	8.2	9.8	9.6	11.5	2.8	3.4	0.26	0.29	7.1	8.2	7.9	9.1	8.9	10.8
Wiki-Talk	2325.5	0.9	1.1	1.4	1.6	4.7	5.8	0.55	0.62	0.1	0.11	1.2	1.5	1.4	1.8	5.6	6.4
LiveJournal	7521.4	4.2	5.3	12.9	15.2	10.1	12.4	1.1	1.6	0.29	0.39	4.6	5.5	12.9	14.8	9.5	11.2



(a) *AGap vs.* $|T|$ (b) Response time *vs.* $|T|$ (c) *DGTwoDIS*: *AGap* (d) *DGTwoDIS*: time efficiency

Fig. 7. Scalability evaluation



(a) *RGap vs.* k (b) Response time *vs.* k

Fig. 6. Effect of the batch size

TABLE IV
EFFECT OF INITIAL INPUT QUALITY (*DGOneDIS* AND *DGTwoDIS*)

Graphs	*DGOneDIS*			*DGTwoDIS*		
	θ=0.50	θ=0.75	θ=0.90	θ=0.50	θ=0.75	θ=0.90
Wiki-Vote	99.91%	99.94%	99.97%	99.97%	99.97%	99.98%
AstroPh	99.86%	99.93%	99.95%	99.95%	99.95%	99.96%
Brightkite	99.46%	99.86%	99.99%	99.99%	99.99%	99.99%
Slashdot	99.94%	99.97%	99.99%	99.99%	99.99%	100%
com-amazon	99.86%	99.94%	99.97%	99.97%	99.99%	99.99%
BerkStan	98.67%	98.93%	99.21%	98.75%	99.33%	99.44%
web-Google	99.73%	99.88%	99.93%	99.91%	99.98%	99.99%
as-skitter	98.96%	99.63%	99.79%	99.67%	99.86%	99.93%
Wiki-Talk	99.91%	99.94%	100%	100%	100%	100%
LiveJournal	99.89%	99.92%	99.96%	99.98%	99.99%	99.99%

4 Conclusions

In this paper, we study the problem of computing the high-quality(may not be the maximum) independent sets over the graphs that are dynamically changing.

Based on the two widely used degree-one and degree-two reduction rules, we devise a very effective index, i.e., dependency graph, that can facilitate the computation of independent sets. The time complexity of searching on a dependency graph is merely $O(d)$ in average case.

Extensive experiments over a wide range of graphs confirm that the proposed methods are both effective and efficient to find high-quality independent sets.