

Compressed representations for web and social graphs

Cecilia Hernandez and Gonzalo Navarro

Presented by Helen Xu

6.886

April 6, 2018

Web graphs and social networks

Web graphs represent the link structure of the Web usually as directed graphs.

Social networks represented relationships among social entities (undirected or directed)

Web graphs and social networks are **growing fast**.

It was recently estimated that the Web was over 7.8 billion pages (around 200 billion edges).

Facebook has over 950 million users.



How do we manage large graphs?

Streaming techniques use main memory and avoid random access to disk [DFR06].

External memory algorithms use memory layouts to exploit locality to reduce I/O cost [V01].

Distributed memory may impose synchronization and communication costs similar to those of the external memory approach [SV11].



How do we manage large graphs?

Streaming techniques use main memory and avoid random access to disk [DFR06].

External memory algorithms use memory layouts to exploit locality to reduce I/O cost [V01].

Distributed memory may impose synchronization and communication costs similar to those of the external memory approach [SV11].

Compressed data structures reduce memory and are often still faster than I/O.



Contributions

1. Extend a technique for detecting bicliques to **detect “dense subgraphs”** [BC08].
2. **Apply “virtual node mining”** to replace edges of the dense subgraph and improve Web graph representation [BC08].
3. **Use a bidirectional representation** (k2-tree) for an improved representation.
4. Introduce a new **compressed data structure** to represent dense subgraphs that does not use virtual nodes.

Compressed representations for Web and social graphs

The **WebGraph framework** exploits power-law distributions, similarity, and locality using URL node ordering [\[BV04\]](#).

Virtual Node Mining (VNM) groups sets of pages that share the same outlinks, which define complete bipartite subgraphs (bicliques) [\[BC08\]](#).

The **k2tree** exploits the sparseness and clustering of the adjacency matrix and supports in/out neighbor queries [\[BLN09\]](#).

And many more! Most can support out-neighbor queries but not necessarily in-neighbor queries.

Bitmaps

Given a bitmap $B[1,n]$,

$\text{rank}(b, i)$ counts the number of times bit b appears in the prefix $B[1, i]$.

$\text{select}(b, i)$ returns the position of the i -th occurrence of bit b in B ($n+1$ if not found)

$\text{access}(i)$ retrieves the value $B[i]$

There exists a compressed bitmap with constant operation times with space $nH_0(B) + o(n)$ bits where $H_0(B) \sim \lg n$ [RRR02]

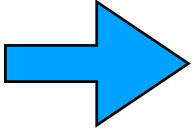
Compact data structures for sequences

Bitmaps can be extended to compact data structures for sequences $S[1, n]$ over an alphabet of size s .

Wavelet trees (WT) supports rank/select/access in $O(\log(s))$ time with $nH_0(S) + o(n)\log(s)$ bits [\[GGV03\]](#).

This paper uses the version for large alphabets that saves extra space $O(s \log(n))$ [\[FG08\]](#).

Contributions

- 
1. Extend a technique for detecting bicliques to **detect “dense subgraphs”** [BC08].
 2. **Apply “virtual node mining”** to replace edges of the dense subgraph and improve Web graph representation [BC08].
 3. **Use a bidirectional representation** (k2-tree) for an improved representation.
 4. Introduce a new **compressed data structure** to represent dense subgraphs that does not use virtual nodes.

Dense subgraph discovery

A Web graph is a directed graph $G = (V, E)$.

For an edge $e = (u, v)$, we say that u is the source and v is the center of e .

Web graphs have “**dense communities**” (a group of pages related to a common interest) characterized by dense directed bipartite subgraphs [RRT19, DGP07].

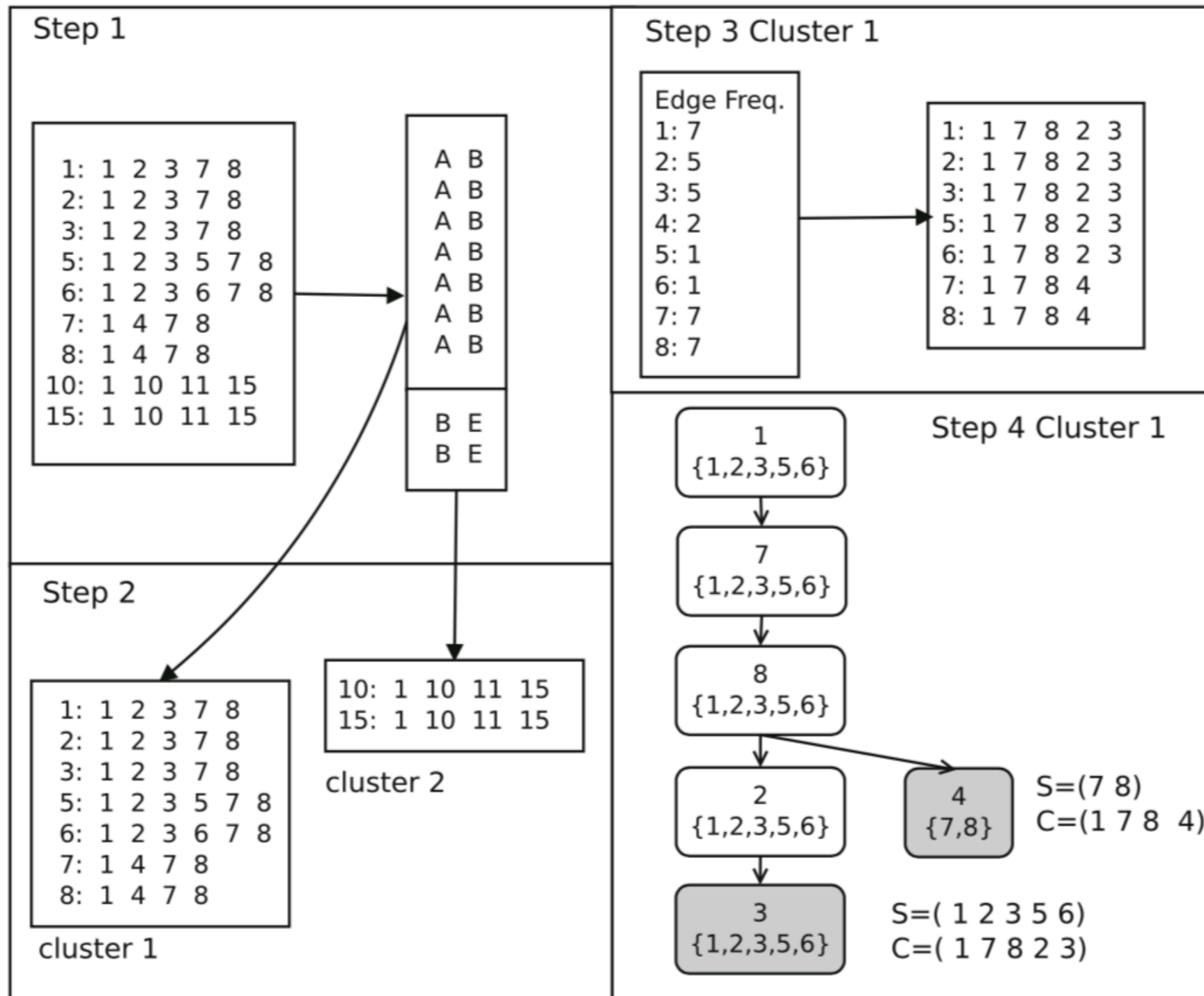
A dense subgraph $H(S, C)$ of graph $G = (V, E)$ is a graph $G'(S \cup C, S \times C)$ where $S, C \subseteq V$.

Algorithm for dense subgraph discovery

1. Clustering-1 - build hashed matrix representing G
2. Clustering-2 - build clusters
3. Mining-1 - reorder cluster edges
4. Mining-2 - discover dense subgraphs and replacing

Total runtime = $O(|E| \log |E|)$

Example: dense subgraph discovery



Evaluation: dense subgraph discovery

Table 1 Compression metrics using different P values with eu-2005

P	# Cliques	Cliques	# Bicliques	Edges	Nodes	Ratio
2	33,482	248,964	58,467	17,208,908	2,357,455	7.30
4	34,237	246,022	60,226	17,199,357	2,426,753	7.08
8	34,863	245,848	60,934	17,205,357	2,524,240	6.81

Evaluation: clique discovery

Table 2 Synthetic clique graphs with different number of nodes (Nodes), edges (Edges), maximum clique size (MC), and total number of vertices participating in cliques (R)

Name	Nodes	Edges	d	MC	R	avg size
PL	999,993	9,994,044	9.99	0	0	—
V16	65,536	610,500	9.31	15	6,548	9.5
V16	65,536	1,276,810	19.48	30	3,785	17.09
V16	65,536	2,161,482	32.98	50	2,398	27.21
V16	65,536	4,329,790	66.06	100	1,263	51.83
V17	131,072	1,214,986	9.26	15	13,130	9.48
V17	131,072	2,542,586	19.39	30	7,589	17.05
V17	131,072	4,309,368	32.87	50	4,790	27.23
V17	131,072	8,739,056	66.67	100	2,495	52.95
V20	1,048,576	9,730,142	9.76	15	104,861	9.50
V20	1,048,576	20,293,364	19.60	30	60,822	17.02
V20	1,048,576	34,344,134	32.90	50	38,544	27.07
V20	1,048,576	69,324,658	66.18	100	20,102	52.10

Column d gives the average number of edges per node, and the last column is the average clique size

Evaluation: Markov Cluster Process

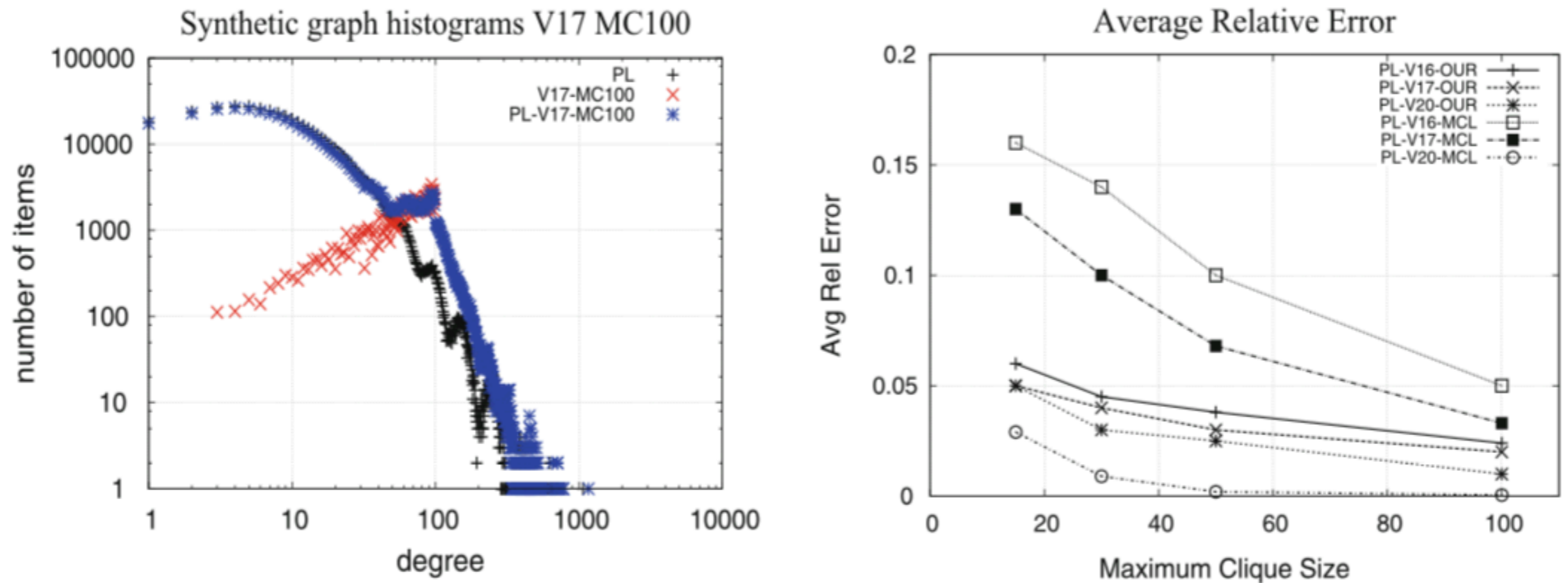


Fig. 3 Outdegree histograms (*left*) and average relative error (*right*) in synthetic graphs

Evaluation: Runtime Comparison

Table 4 Time required per retrieved clique of different sizes

Name	MC	$ A $	avg	tms	$ A M$	$avgM$	$tmsM$	$ptmsM$
PL-V16	15	6,501	9.00	236.1	5,810	7.96	4,359.2	1,938.5
PL-V16	30	3,766	16.53	336.4	3,596	15.18	7,877.3	3,129.1
PL-V16	50	2,389	26.58	305.1	2,331	25.40	11,190.4	5,089.2
PL-V16	100	1,261	51.08	590.0	1,242	50.80	19,839.7	9,363.1
PL-V17	15	13,071	9.00	120.5	12,032	8.30	2,048.4	977.9
PL-V17	30	7,565	16.53	129.8	7,321	15.83	3,226.3	1,612.3
PL-V17	50	4,776	26.70	203.1	4,706	26.21	4,886.3	2,394.1
PL-V17	100	2,492	51.85	318.2	2,481	51.89	10,153.5	4,446.1
PL-V20	15	104,771	9.06	103.1	103,437	9.31	580.2	103.6
PL-V20	30	60,773	16.56	150.3	60,614	16.97	614.6	152.4
PL-V20	50	38,524	26.62	155.4	38,473	27.09	639.7	248.2
PL-V20	100	20,095	51.62	178.6	20,097	52.11	1,371.1	505.7

Summary of evaluation

Relative error values are low in this hashing approach, whereas the error grows with MCL when the graph contains smaller or fewer cliques.

This clustering algorithm has high discovery rates (over 93%) for various graph structures, while MCL is sensitive to the number and size of cliques (and is less effective for fewer or smaller cliques)

MCL has scalability problems and performs poorly on sparse graphs [\[MSAK11\]](#) and takes $O(V^3)$ time.

Contributions

1. Extend a technique for detecting bicliques to **detect “dense subgraphs”** [BC08].

 2. **Apply “virtual node mining”** to replace edges of the dense subgraph and improve Web graph representation [BC08].

3. **Use a bidirectional representation** (k2-tree) for an improved representation.

4. Introduce a new **compressed data structure** to represent dense subgraphs that does not use virtual nodes.

Dense subgraph mining

while (new dense subgraphs found) {

1. Use dense subgraph discovery from previous slides.

2. Apply virtual nodes on original graph to factor out edges of discovered dense subgraphs.

}

Use compression techniques and node orderings on the output graph.

Evaluation: dense subgraph mining

Table 8 Main statistics on the *DSM* reduced graphs

Dataset	T	$ V_3 $	$ E_3 $	d_3	$ E_2 / E_3 $	$ VN $	ET (min)
eu-2005	10	1,042,260	3,516,473	3.37	5.32	179,596	3.45
	5	1,019,699	3,776,194	3.70	4.96	157,035	2.45
indochina-2004	10	8,079,568	21,313,402	2.63	8.99	664,703	35.0
	5	8,030,729	22,186,260	2.76	8.63	615,864	24.3
uk-2002	10	19,842,886	54,391,059	2.74	5.37	1,322,400	65.8
	5	19,767,439	56,329,408	2.84	5.18	1,246,953	44.2
arabic-2005	10	26,193,219	74,071,714	2.82	8.52	3,449,139	185.1
	5	25,805,521	78,919,645	3.05	7.99	3,061,441	130.3

Dense subgraph mining

while (new dense subgraphs found) {

1. Use dense subgraph discovery from previous slides.

2. Apply virtual nodes on original graph to factor out edges of discovered dense subgraphs.

}

Use compression techniques and node orderings on the output graph.

Performance evaluation with out-neighbor support

The authors compared DSM with the best alternatives BV [BRSV11], AD [AD09], and GB [GB11].

Table 9 Compression performance in bpe, with support for out-neighbor queries

Dataset	eu-2005	indochina-2004	uk-2002	arabic-2005
BV_{m100w7}	3.74	1.50	2.38	1.79
AD_8	3.64	1.60	2.64	2.26
GB_{128}	1.83	<i>1.09</i>	1.76	1.35
DSM+ES _x -T10+BV	3.06	1.48	2.68	2.06
DSM-ES _x -T5+AD ₄	2.44	1.18	2.05	1.56
DSM-ES _x -T5+AD ₈	2.30	1.06	1.87	1.45
DSM-ES _x -T10+AD ₄	2.32	1.14	2.01	1.51
DSM-ES _x -T10+AD ₈	2.20	1.03	<i>1.83</i>	<i>1.40</i>

The best-performing one per graph is in bold and the second best in italics

Summary: space/time tradeoffs

Both BV and AD improved when combined with DSM. GB dominates all the others besides the combination.

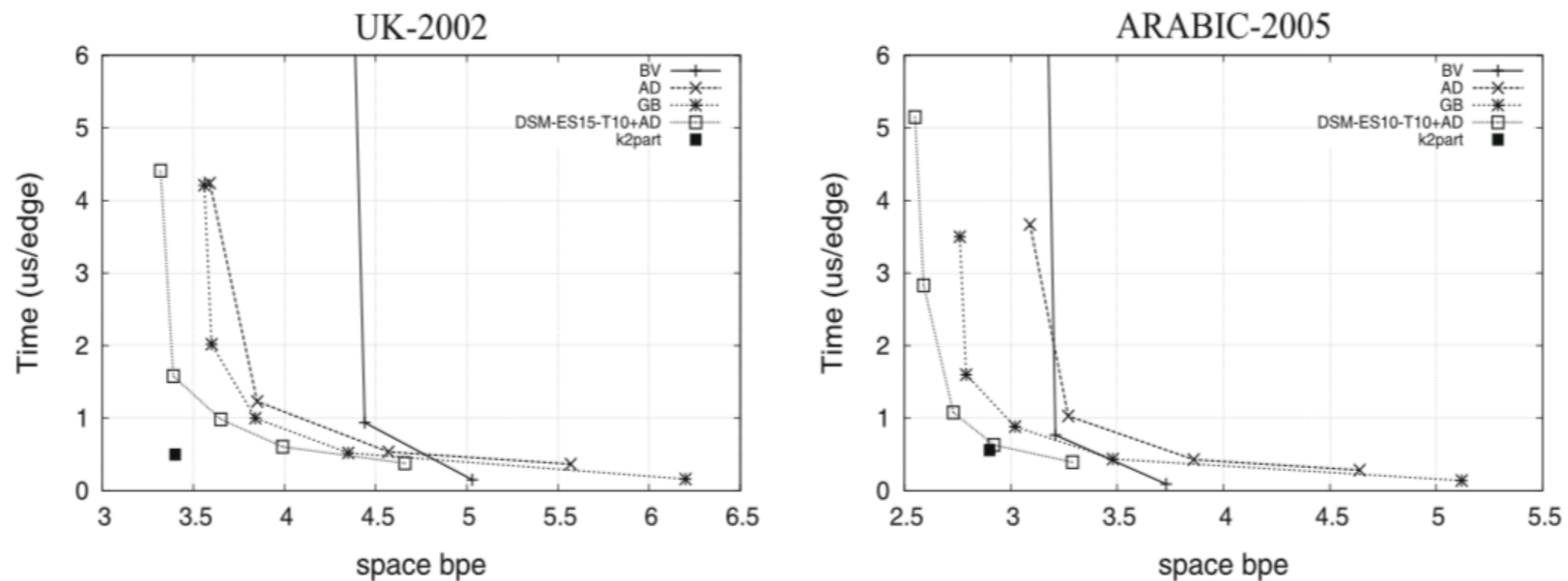
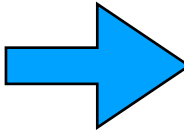


Fig. 6 Space/time efficiency with out/in-neighbor queries

Contributions

1. Extend a technique for detecting bicliques to **detect “dense subgraphs”** [BC08].
2. **Apply “virtual node mining”** to replace edges of the dense subgraph and improve Web graph representation [BC08].
-  3. **Use a bidirectional representation** (k2-tree) for an improved representation.
4. Introduce a new **compressed data structure** to represent dense subgraphs that does not use virtual nodes.

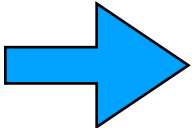
Performance evaluation with out/in-neighbor support

The authors combined the output of DSM with k2-trees, a compression technique that supports out/in-neighbor queries [BLN12]

Table 10 Compression performance when combining with *k2trees*

Dataset	eu-2005	indochina-2004	uk-2002	arabic-2005
k2treeNAT	3.45	1.35	2.77	2.47
k2treeBFS	3.22	1.23	2.04	1.67
DSM-ES10-T5 + k2treeNAT	2.76	1.36	2.40	1.76
DSM-ES10-T10 + k2treeNAT	2.71	1.34	2.40	1.76
DSM-ES15-T5 + k2treeNAT	2.65	1.27	2.28	1.67
DSM-ES15-T10 + k2treeNAT	2.59	1.27	2.27	1.66
DSM-ES100-T5 + k2treeNAT	2.56	1.16	2.13	1.52
DSM-ES100-T10 + k2treeNAT	2.48	1.14	2.08	1.47
DSM-ES10-T5 + k2treeBFS	2.21	0.90	1.56	1.12
DSM-ES10-T10 + k2treeBFS	2.11	0.87	1.53	1.08
DSM-ES15-T5 + k2treeBFS	2.11	0.87	1.54	1.14
DSM-ES15-T10 + k2treeBFS	2.21	0.89	1.57	1.08
DSM-ES100-T5 + k2treeBFS	2.54	0.95	1.67	1.21
DSM-ES100-T10 + k2treeBFS	2.45	0.93	1.64	1.18

Contributions

1. Extend a technique for detecting bicliques to **detect “dense subgraphs”** [BC08].
2. **Apply “virtual node mining”** to replace edges of the dense subgraph and improve Web graph representation [BC08].
3. **Use a bidirectional representation** (k2-tree) for an improved representation.
-  4. Introduce a new **compressed data structure** to represent dense subgraphs that does not use virtual nodes.

Compact data structure for dense subgraphs

Extract dense subgraphs and represent them using compact data structures based on bitmaps.

Algorithm 1: Construction of X and B

Input: Subsets S_1, \dots, S_N and C_1, \dots, C_N

Output: Sequence X and Bitmap B

$X \leftarrow \varepsilon;$

$B \leftarrow \varepsilon;$

for $i \leftarrow 0$ **to** N **do**

$L \leftarrow S_i - C_i;$

$M \leftarrow S_i \cap C_i;$

$R \leftarrow C_i - S_i;$

$X \leftarrow X : L : M : R;$

$B \leftarrow B : 10^{|L|} 10^{|M|} 10^{|R|};$

end

return $X, B;$

(a) Pattern extraction

Input graph (sorted lists)

1:	1	2	3	7	8	
2:	1	2	3	7	8	
3:	1	2	3	7	8	
5:	1	2	3	5	7	8
6:	1	2	3	6	7	8

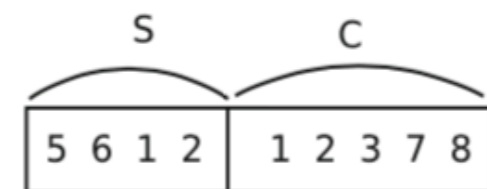
pattern

1:	1	2	3	7	8
2:	1	2	3	7	8
3:	1	2	3	7	8
5:	1	2	3	7	8
6:	1	2	3	7	8

remaining

5:	5
6:	6

(b) Our compressed representation



X

5	6	1	2	3	7	8
---	---	---	---	---	---	---

B

1	0	0	1	0	0	0	1	0	0
---	---	---	---	---	---	---	---	---	---

Results summary

Modest space gains on social networks and no space gains on Web graphs.

This approach is **dominated in space and time** by previously proposed compression techniques, but **can answer various mining queries** (e.g. density) related to dense subgraphs easily and without extra space.

Conclusions

Dense-subgraph-mining-based approaches provide the best time while using little space out of techniques that provide in- and out-neighbor queries.

When combined with k2trees, the result is the most space-efficient representation of Web graphs.

The compression scheme presented is better for social networks with out- and in-neighbor support.