Graph Mining Techniques
HDB-Subdue

Sharma Chakravarthy
Information Technology Laboratory
Computer Science and Engineering Department
The University of Texas at Arlington, Arlington, TX 76009
Email: sharma@cse.uta.edu
URL: http://itlab.uta.edu/sharma
Course URL: http://wweb.uta.edu/faculty/sharmac

Acknowledgments

- Parts of this presentation are based on the work of many of my students, especially Ramji Beera, Ramanathan Balachandran, Srihari Padmanabhan, Subhesh Pradhan (and others)
- National Science Foundation and other agencies for their support of MavHome, Graph mining and other projects
- Some slides are borrowed from various sources (web and others)

Tutorial Outline

- ***Graph Mining Approaches***
  - Subdue
  - AGM
  - *FSG*
- ***SQL-Based Graph Mining***
  - *HDB-Subdue*
  - *DB-FSG (may be)*
- ***Graph mining applications***
  - Email classification
- Multilayer Network Analysis
- Conclusions
- References

Conclusions

- Graph mining is a powerful approach needed by many real-world applications
- There is need for both Subdue class of mining algorithms and frequent subgraph class of mining algorithms
- Scalability is an extremely important issue
- Our approach to using SQL has yielded very promising scalability results (800K vertices and 1600K edges)
- Our approach using map/reduce has mapped the algorithm to Map/reduce, further optimized it, and made it scalable to any data size!

Comparison

	Subdue	FSG	AGM	gSpan	HDBSubdue
Graph Mining	✓	✓	✓	✓	✓
Multiple edges	✓	×	×	×	✓
Hierarchical reduction	✓	×	×	×	✓
Cycles	✓	✓	✓	×	✓
Evaluation metric	MDL	Frequency	Support, Confidence	Frequency	DMDL (frequency)
Inexact graph match With threshold	✓	×	×	×	×
Memory limitation	✓	✓	✓	✓	×

Scalability Issues

- Subdue is a main memory algorithm.
- Good performance for small data sizes
- Entire graph is constructed before applying the mining algorithm
- Takes a very long time to even to initialize for 1600K edges and 800K vertices graph
- Scalability is an issue

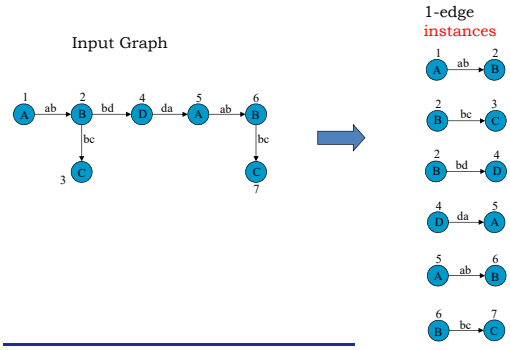
SQL-Based Graph Mining

- We have mapped the Subdue algorithm using SQL (HDB-Subdue)
  - Handles multiple edges between nodes
  - Handles cycles/loops
  - Performs Hierarchical reduction
  - Dveloped DMDL tailored to databases
- Can handle graphs of Millions of edges and vertices
- DB-FSG does frequent subgraph mining
- Working on inexact matching

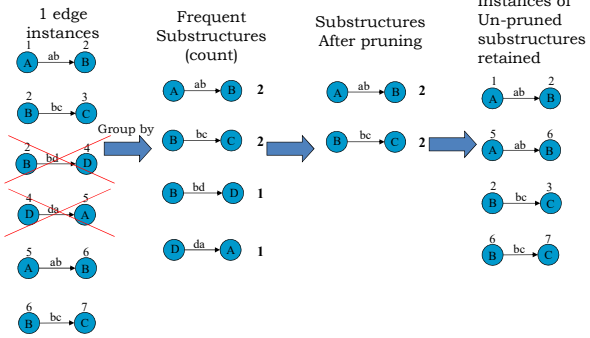
HDB-Subdue

- Graph representation using relations
- Joins used for iterative generation of larger substructures
- Pseudo duplicate elimination involves a number of joins
- DMDL is used to identify the best substructure (count/frequency can be used as well)

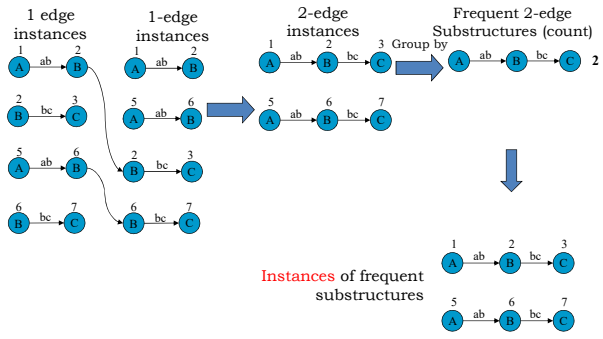
HDB-Subdue



1 edge pruning



Generating 2 edge substructures



Representing input graph using relations

Input stored in two tables

Vertices

vertexno	vertexlabel
1	A

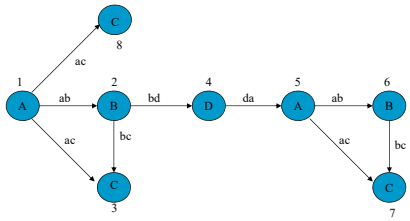
Edges

vertex1	vertex2	edgename
1	2	ab

The pseudo code of HDB-Subdue

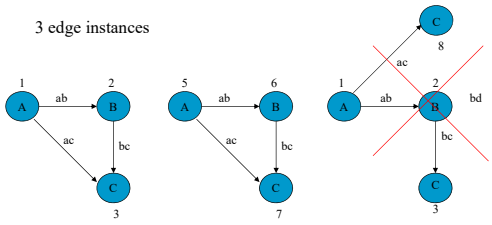
INPUT : (Input graph flat file, MaxSize, Beam, NumberOfIterations)
BEGIN HDB-Subdue
  Load vertices and edges from flat file into relations;
  Remove the edges which have only one instance from the edge table
  For (i=1 to NumberOfIterations) { // For Hierarchical reduction
    For (j=2 to MaxSize) { // Substructure expansion
      Join instance_(j-1) with oneedge table to generate instance_j
      Eliminate pseudo duplicates from instance_j table
        Group instance_j by vertex label and edge label to obtain substructures and its count
      Calculate DMDL for substructures and insert them into sub_fold_j table
        Retain top Beam substructures in sub_fold_j table and remove the rest
      Retain only the instances of sub_fold_j table in instance_j table and eliminate others
    }
    --- End of iteration i ---
  Select the best substructure of iteration i
  Compress the substructure by updating vertex table and oneedge table
  }
End HDB-Subdue

Representation of a Substructure



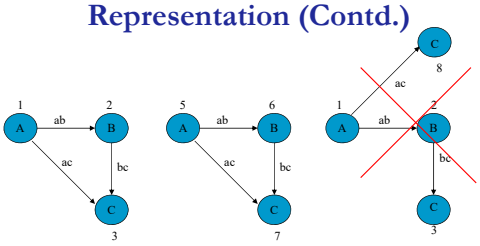
Representation (Contd.)

3 edge instances



Isomorphic instance Count 3

Representation (Contd.)



HDB instance_3 (instances of size 3)

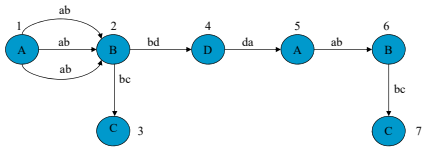
V1	V2	V1	V3	V2	V4	V1L	V2L	V3L	V4L	E1	E2	E3	F1	T1	F2	T2	F3	T3
1	2	1	3	2	0	A	B	C	-	ab	bc	ac	bd	ac	2	3	1	3
5	6	5	7	6	0	A	B	C	-	ab	bc	ac	bd	ac	2	3	1	3
1	2	1	3	2	8	A	B	C	C	ab	bc	ac	bd	ac	2	3	1	4

Count 2

Representation

- To represent an n-edge substructure, we need 6n+2 attributes
  - n+1 vertex numbers
  - n+1 vertex labels
  - n edge numbers
  - n edge labels, and
  - n from relative vertex id
  - n to relative vertex id
- Given that databases support about 1000 attributes now, the max substructure that can be represented is about 165 edges

Need for Edge numbers



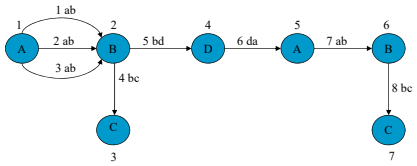
vertex1	vertex2	edge1	vertex1name	vertex2name
1	2	ab	A	B
1	2	ab	A	B
1	2	ab	A	B
.	.	.	.	.

EDB-Oneedge table

vertex1	vertex2	edge No	edge1	vertex1name	vertex2name
1	2	1	ab	A	B
1	2	2	ab	A	B
1	2	3	ab	A	B
.	.	.	.	.	.

HDB-Oneedge table

Constrained Expansion



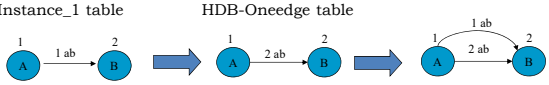
For each iteration, number of queries will increase for expansion as the number of vertices increases

They have to be generated correctly!

Unconstrained expansion

Instance_1 table

HDB-Oneedge table



vertex1	vertex2	edge1No	edge1	vertex1name	vertex2name
1	2	1	ab	A	B
.	.	.	.	.	.

Instance_1 table

=

vertex1	vertex2	edgeNo	edge	vertex1name	vertex2name
1	2	2	ab	A	B
.	.	.	.	.	.

HDB-Oneedge table

WHERE i.vertex1=o.vertex1 and o.edgeNo <> i.edge1No

Expansion

- To expand a 1-edge substructure with 2 vertices V1 and V2, we need to:
  - expand self-edges on V1 or V2 (in general n)
  - expand multiple edge from v2 to v1 (or n*(n-1))
  - expand incoming edge on V1 or V2 (or n)
  - expand outgoing edge on V1 or V2 (or n)
- In general, to expand a substructure of size n, we need n**2 + 2n queries
- These queries are generated (can be one union query)
- If you know that multiple edges and/or cycles do not exist, less number of queries can be used

Expansion SQL query Example

```
/* Query to expand on V1 on a outgoing edge from V1 to new vertex V3 */
INSERT INTO instance_2 (
    SELECT s.vertex1, s.vertex2, o.vertex2, s.vertex1name,  s.vertex2name,
        o.vertex2name, s.edge1, o.edge, s.edge1name, o.edgename,
        s.from_1, s.to_1, 1, 3
    FROM InstanceIter_1 s, oneedge o
        WHERE o.vertex1=s.vertex1 and o.edge<>s.edge1 and
        o.vertex2<>s.vertex1 and o.vertex2<>s.vertex2)
Similarly,
/* Query to expand self edge on vertex 1 */
/* Query to expand self edge on vertex 2 */
/* Query to expand multiple edge from V1 to V2*/
/* Query to expand multiple edge from V2 to V1*/
/* Query to expand on V1 on a outgoing edge from V1 to new vertex V3 */
/* Query to expand on V2 to new vertex V3 */
/* Query to expand incoming edge on V1 from new vertex V3 */
/* Query to expand incoming edge on V2 from new vertex V3 */
```


Expansion

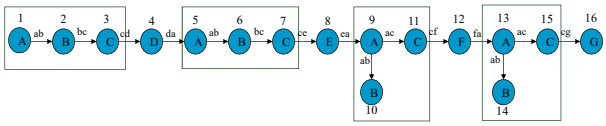
VL1	VL2	VL3	EL1	EL2	F1	T1	F2	T2	COUNT	DMDL
A	B	C	AB	AC	1	2	1	3	3	1.8
A	B	D	AB	BD	1	2	2	3	1	0.9
B	D	A	BD	DA	1	2	2	3	1	0.9
D	A	B	DA	AB	1	2	2	3	1	0.9
D	A	C	DA	AC	1	2	2	3	1	0.9

The pseudo code of HDB algorithm

```
INPUT : (Input graph flat file, MaxSize, Beam, NumberOfIterations)
BEGIN HDB-Subdue
  Load vertices and edges from flat file into relations;
  Remove the edges which have only one instance from the edge table
  For (i=1 to NumberOfIterations) { // For Hierarchical reduction
    For (j=2 to MaxSize) { // Substructure expansion
      Join instance_(j-1) with oneedge table to generate instance_j
      Eliminate pseudo duplicates from instance_j table
        Group instance_j by vertex label and edge label to obtain substructures and its
      count
      Calculate DMDL for substructures and insert them into sub_fold_j table
        Retain top Beam substructures in sub_fold_j table and remove rest
      Retain only the instances of sub_fold_j table in instance_j table and eliminate
      others
    }
    --- End of iteration i ---
    Select the best substructure of iteration i
    Compress the substructure by updating vertex table and oneedge table
  }
End HDB-Subdue
```

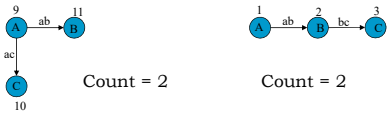
Substructure Evaluation metric

Example



2 Edge substructure instances

Subdue MDL



Count = 2 Count = 2

$$\text{MDL} = \frac{\text{DL(G)}}{\text{DL(S)} + \text{DL(G|S)}}$$

	9	10	11
9	0	1	1
10	0	0	0
11	0	0	0

Adjacency matrix

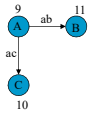
MDL = 1.14539

	1	2	3
1	0	1	0
2	0	0	1
3	0	0	0

Adjacency matrix

MDL = 1.12849

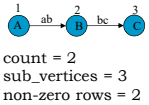
DMDL



count = 2
sub_vertices = 3
non-zero rows = 1

	9	10	11
9	0	1	1
10	0	0	0
11	0	0	0

DMDL = 1.1481



count = 2
sub_vertices = 3
non-zero rows = 2

	1	2	3
1	0	1	0
2	0	0	1
3	0	0	0

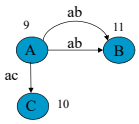
DMDL = 1.1071

$$\text{DMDL} = \frac{\text{Value (G)}}{\text{Value (S)} + \text{Value (G|S)}}$$

- Value(G) = graph_vertices + graph_edges
- Value(S) = sub_vertices + non-zero rows (out degree of at least 1)
- Value(G|S) = (graph_vertices – sub_vertices * count + count) + (graph_edges – sub_edges * count)

sub_vertices

Three edge



EDB – Subdue:

No of vertices = No of edges + 1

~~No of vertices = 3 + 1 = 4~~

V1	V2	V3	V4	V1L	V2L	V3L	V4L	E1	E2	E3	F1	T1	F2	T2	F3	T3
9	**11**	**10**	**0**	A	B	-	-	AB	AB	AC	**1**	**2**	**1**	**2**	**1**	**3**
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

Instance_3 table

Repeating vertex marked by vertex invariant (0's and –'s)

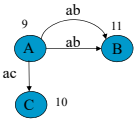
No of vertices = count of non zero vertex numbers = 3

OR

No of vertices = count of unique connectivity attributes = 3

non-zero rows

Three edge



	9	10	11
9	0	1	1
10	0	0	0
11	0	0	0

Adjacency matrix

Non Zero Rows = 1

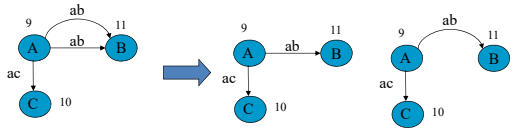
Instance_3 table

V1	V2	V3	V3	V1L	V2L	V3L	V3L	E1	E2	E3	F1	T1	F2	T2	F3	T3
9	11	10	0	A	B	-	-	ab	ab	ac	1	2	1	2	1	3
.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.

- Non zero row in adjacency matrix will have out degree of at least 1
- The Fn attribute represents outgoing vertex
- Count of unique Fn attributes gives number of non-zero rows

count

Two edge substructure instances



Count = 2

- Value(G|S) = (graph_vertices – sub_vertices * count + count) + (graph_edges – sub_edges * count)
  = (3 – 3*2 + 2) + (3 – 2*2) = -2
  = (3 – 3*1 + 1) + (3 – 2*1) = 2
- Tells how many vertices of the entire graph, the substructure can compress
- Both the instances remove same set of vertices, therefore count = 1

count (Contd..)

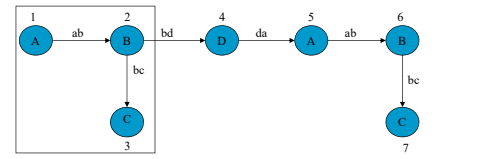
Instance_2 table

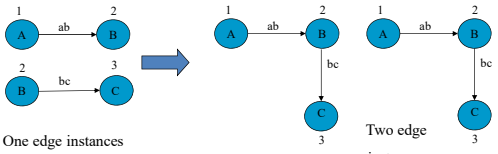
V1	V2	V3	V1L	V2L	V3L	E1	E2	F1	T1	F2	T2
9	11	10	A	B	C	ab	ac	1	2	1	3
9	11	10	A	B	C	ab	ac	1	2	1	3
.	.	.	.	.	.	.	.	.	.	.	.

Count = 2
1 Group with 2 instances

- GROUP BY vertex, edge label and connectivity attributes gives count of all instances, i.e. 2
- GROUP BY vertex number, within instances of a particular substructure will return group of instances containing unique vertex numbers
- Count on the number of such groups gives the updated count of instances. In this case it is 1
- Therefore Instead of counting 2, we count as 1

Pseudo duplicates

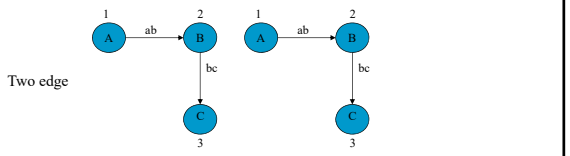




One edge instances

Two edge instances

Pseudo duplicates (Contd..)



Instance_2 table

V1	V2	V3	V1L	V2L	V3L	E1	E2	F1	T1	F2	T2
1	2	3	A	B	C	AB	BC	1	2	2	3
1	2	3	A	B	C	AB	BC	1	2	2	3
.	.	.	.	.	.	.	.	.	.	.	.

Pseudo duplicates (Contd..)

In SQL, only rows can be sorted

Tasks:

- Convert **columns into rows**
- **Sort** the **rows**
- Convert **rows into columns**

Instance_2 table

V1	V2	V3	V1L	V2L	V3L	E1	E2	F1	T1	F2	T2
2	3	1	B	C	A	BC	AB	1	2	3	1
.	.	.	.	.	.	.	.	.	.	.	.

Unsorted_V

V	VL	POS
2	B	1
3	C	2
1	A	3
.	.	.

E	F	T
BC	1	2
AB	3	1
.	.	.

Unsorted_E

Updating connectivity attributes

Unsorted_V

V	VL	POS
2	B	1
3	C	2
1	A	3
.	.	.

Sort on V

Sorted_V

V	VL	POS	NEW POS
1	A	3	1
2	B	1	2
3	C	2	3
.	.	.	.

3 Way Join

Sorted_V

V	VL	POS	NEW POS
1	A	3	1
2	B	1	2
3	C	2	3
.	.	.	.

Unsorted_E

E	F	T
BC	1	2
AB	3	1
.	.	.

Sorted_V

V	VL	POS	NEW POS
1	A	3	1
2	B	1	2
3	C	2	3
.	.	.	.

Updated_E

E	F	T
BC	2	3
AB	1	2
.	.	.

Sort on F, T

Sorted_E

E	F	T
AB	1	2
BC	2	3
.	.	.

Reconstructing instance table

Sorted_V

V	VL	POS	NEW POS
1	A	3	1
2	B	1	2
3	C	2	3
.	.	.	.

Sorted_V

V	VL	POS	NEW POS
1	A	3	1
2	B	1	2
3	C	2	3
.	.	.	.

Sorted_V

V	VL	POS	NEW POS
1	A	3	1
2	B	1	2
3	C	2	3
.	.	.	.

Sorted_E

E	F	T
AB	1	2
BC	2	3
.	.	.

Sorted_E

E	F	T
AB	1	2
BC	2	3
.	.	.

2n+1 Way Join for reconstruction

Instance_2 table

V1	V2	V3	V1L	V2L	V3L	E1	E2	F1	T1	F2	T2
1	2	3	A	B	C	AB	BC	1	2	2	3
.	.	.	.	.	.	.	.	.	.	.	.

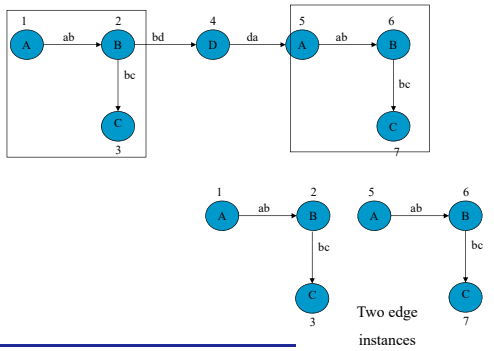
Pseudo duplicates (Contd..)

- Group By Vertex numbers and edge direction attributes
- Retain one and eliminate other

V1	V2	V3	V1L	V2L	V3L	E1	E2	F1	T1	F2	T2
1	2	3	A	B	C	AB	BC	1	2	2	3
1	2	3	A	B	C	AB	BC	1	2	2	3
.	.	.	.	.	.	.	.	.	.	.	.

Instance_2 table

Canonical label ordering

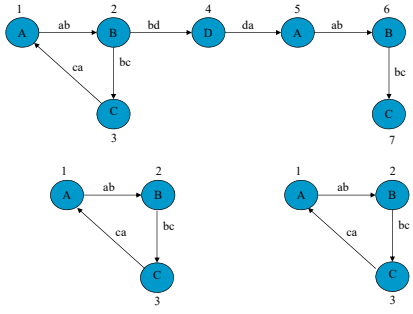


Two edge instances

Canonical label ordering

- These two are NOT duplicates
- They need to be recognized as isomorphic to each other
- In order to do this, vertices and labels need to be canonically ordered
- This process for canonical ordering is similar to the process used for psuedo duplicate elimination

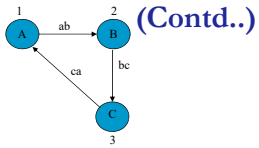
Cycles – Marking repeated vertex



Three edge starting with ab

Three edge starting with bc

Cycles – Marking repeated vertex (Contd..)



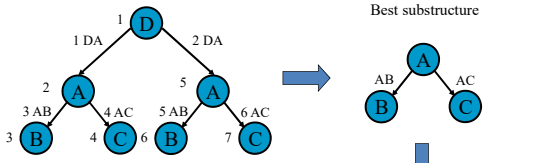
V1	V2	V3	V4	V1L	V2L	V3L	V4L	E1	E2	E3	F1	T1	F2	T2	F3	T3
1	2	3	1	A	B	C	A	AB	BC	CA	1	2	2	3	3	4
2	3	1	2	B	C	A	B	BC	CA	AB	1	2	2	3	3	4

Instance_3 table

V1	V2	V3	V4	V1L	V2L	V3L	V4L	E1	E2	E3	F1	T1	F2	T2	F3	T3
0	2	3	0	A	B	C	-	AB	BC	CA	2	3	3	4	4	2
0	3	2	0	B	C	B	-	BC	CA	CB	2	3	3	4	4	2

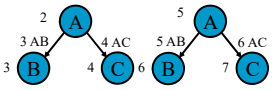
Instance_3 table with vertex invariants 0's and –'s

Hierarchical Reduction



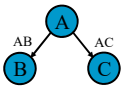
Input graph

Best substructure



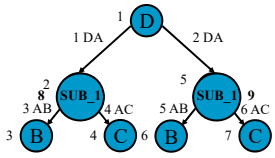
Best substructure instances

Hierarchical Reduction (Contd..)



Best substructure

Compressing vertex



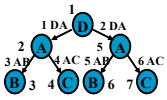
Input graph

Hierarchical Reduction (Contd..)

Tasks in Hierarchical Reduction:

1. Select the best substructure after each iteration
2. Identify the instances of the best substructure
3. For each instance
   - (1) Remove the vertices from the input graph
   - (2) For every instance removed include a new vertex to the input graph
   - (3) Remove the edges from the input graph
   - (4) Update the vertex number of the edges that are incident on or going out of the compressed instance
4. The compressed input graph participates in the next iteration

Selecting the best substructure

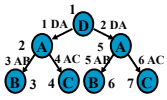


subfold_2

V1L	V2L	V3L	E1	E2	F1	T1	F2	T2	COUNT	DMDL
A	B	C	AB	AC	1	2	1	3	2	1.71
D	A	B	DA	AB	1	2	2	3	2	1.34
D	A	C	DA	AC	1	2	2	3	2	1.34
D	A	A	DA	DA	1	2	1	3	1	0.75

Best substructure is one with highest DMDL value

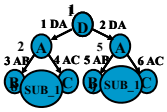
Identify instances of best substructure



V1	V2	V3	V1L	V2L	V3L	E1N	E2N	E1	E2	F1	T1	F2	T2	DMDL
2	3	4	A	B	C	3	4	AB	AC	1	2	1	3	1.71
5	6	7	A	B	C	5	6	AB	AC	1	2	1	3	1.71

BestInstances

Updating vertex table



V1	V2	V3	V1L	V2L	V3L	E1N	E2N	E1	E2	F1	T1	F2	T2	DMDL
2	3	4	A	B	C	3	4	AB	AC	1	2	1	3	1.71
5	6	7	A	B	C	5	6	AB	AC	1	2	1	3	1.71

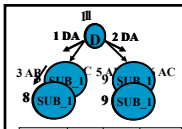
BestInstances

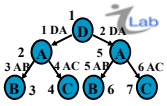
```
DELETE FROM VertexTable
WHERE VertexNo IN (
        (SELECT V1 FROM BestInstances)
UNION (SELECT V2 FROM BestInstances)
                . . . .
UNION (SELECT Vn+1 FROM BestInstances))
```

Vertex table

V	VL	VL
1	D	D
2	A	SUB_1
3	B	SUB_1
4	C	
5	A	
6	B	
7	C	

Updating oneedge table





V1	V2	V3	V1L	V2L	V3L	E1N	E2N	E1	E2	F1	T1	F2	T2	DMDL
2	3	4	A	B	C	3	4	AB	AC	1	2	1	3	1.71
5	6	7	A	B	C	5	6	AB	AC	1	2	1	3	1.71

BestInstances

```
UPDATE oneedge set V1=MAX_VERTEX
WHERE V1 IN (
        (SELECT V1 FROM BestInstances WHERE rownum = 1)
UNION (SELECT E2N FROM BestInstances)
UNION (SELECT Vn+1 FROM BestInstances WHERE rownum = 1))
                . . . .
UNION (SELECT EnN FROM BestInstances))
UPDATE oneedge set V2=MAX_VERTEX
WHERE V2 IN (
        (SELECT V1 FROM BestInstances WHERE rownum = 1)
                . . . .
UNION (SELECT Vn+1 FROM BestInstances WHERE rownum = 1))
```

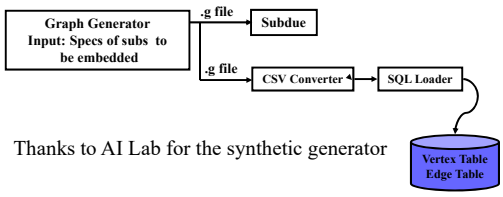
Oneedge table

EL	EN	V1L	V2L	V1	V2
DA	1	D	SUB_1	1	2 8
DA	2	D	SUB_1	1	5 9
AB	3	A	B	2	3
AC	4	A	C	2	4
AB	5	A	B	5	6
AC	6	A	C	5	7

Experimental Results

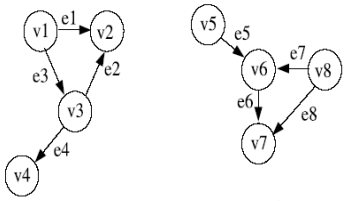
Setup:

- Input graphs generated using the graph generator developed by AI Lab
- Platform: Linux
- Database: Oracle 10g
- Machine's memory: 2 Gbytes
- Number of processors: 2



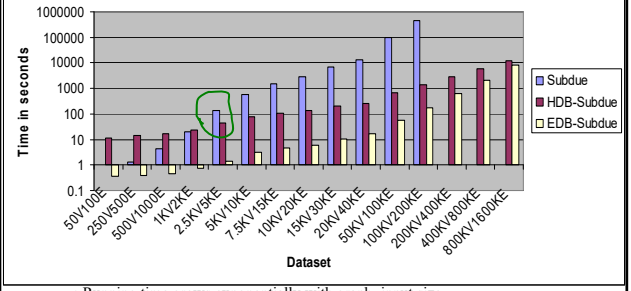
Thanks to AI Lab for the synthetic generator

No cycles and multiple edges



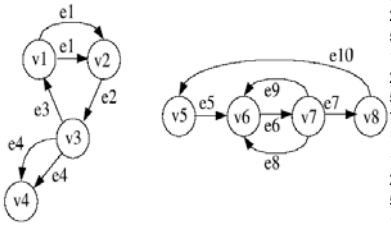
Dataset	Instances
50V100E	4
250V500E	15
500V1000E	30
1KV2KE	60
2.5KV5KE	150
5KV10KE	300
7.5KV15KE	450
10KV20KE	600
15KV30KE	900
20KV40KE	1200
50KV100KE	3000
100KV200KE	6000
200KV400KE	12000
400KV800KE	24000
800KV1600KE	48000

No cycles and multiple edges



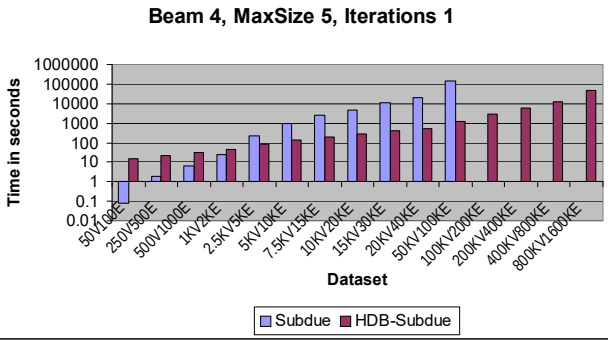
- Running time grows exponentially with graph input size
- Subdue crossover – 2.5KV5KE

With cycles and multiple edges



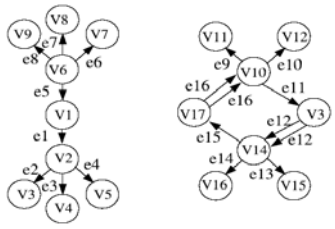
Dataset	Instances
50V100E	4
250V500E	15
500V1000E	30
1KV2KE	60
2.5KV5KE	150
5KV10KE	300
7.5KV15KE	450
10KV20KE	600
15KV30KE	900
20KV40KE	1200
50KV100KE	3000
100KV200KE	6000
200KV400KE	12000
400KV800KE	24000
800KV1600KE	48000

With cycles and multiple edges



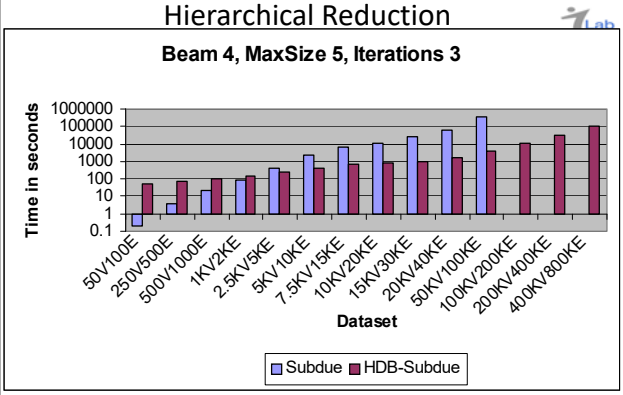
• Subdue crossover – 2.5KV5KE

Hierarchical Reduction



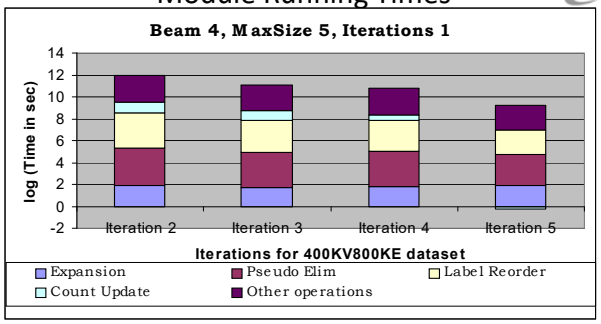
Dataset	Instances
50V100E	2
250V500E	10
500V1000E	20
1KV2KE	40
2.5KV5KE	80
5KV10KE	160
7.5KV15KE	250
10KV20KE	320
15KV30KE	420
20KV40KE	640
50KV100KE	1280
100KV200KE	2560
200KV400KE	5120
400KV800KE	10240
800KV1600KE	20480

Hierarchical Reduction



• Subdue crossover – 2.5KV5KE

Module Running Times



• Pseudo duplicate elimination takes most of the time because of 2n+1 way join

• Expansion time decreases with iteration as the number of tuples handled in each iteration becomes less

Conclusion

- Captured complete graph information using new schema
- Unconstrained expansion can expand multiple edges and a scheme for eliminating pseudo duplicates was provided
- DMDL has been modified to account for multiple edges
- Handles Cycles in the input graph
- Performs hierarchical graph reduction
- Tested to mine input graphs of size 800KV1600KE

DB-FSG

- DB-FSG is a relational database approach for frequent subgraph mining.
- Addresses scalability issues much better than the main memory algorithm.
- It uses database relations to represent a graph
- Steps of DB-FSG
  - Candidate Generation
  - Frequency counting
  - Sub-graph pruning

Comparison chart of FSG and DB-FSG

	FSG	DB-FSG
Sub-graphs Representation	Sparse Adjacency Matrix Graphs without cycles and multiple edges	Tuples of instance_n relation represents subgraphs of size n Can handle graphs with cycles and multiple edges
Candidate Generation	Joins two frequent subgraphs of size n that has same core of size n-1 to generate size n+1 candidate subgraphs Uses canonical labeling to avoid duplicates Uses downward closure property to retain true candidates	Perform SQL join of instance_n relation and one_edge relation to generate candidate subgraphs of size n+1 Uses edge code approach for removing pseudo duplicates.
Frequency Counting	Uses canonical labeling for subgraph isomorphism and frequency counting	Uses Canonical ordering on vertex labels and group by function for frequency counting