T-Trees, Vertical Partitioning and Distributed Association Rule Mining

Frans Coenen, Paul Leng and Shakil Ahmed

Department of Computer Science, The University of Liverpool

Liverpool, L69 3BX, UK

{frans,phl,shakil}@csc.liv.ac.uk

Overview and Motivation

- An approach to distributed/parallel ARM (**DATA-VP**) is presented that makes use of a vertical partitioning strategy to distribute the input data set.
- Features:
  1. Founded on a compressed set enumeration tree (the **T-tree**) together with an associated ARM algorithm (**Apriori-T**).
  2. Partitions can be mined in isolation.
  3. Partitioning is such that the possibility of the existence of large itemsets dispersed across partitions is taken into account.

The Total Support <u>T</u>ree (<u>T</u>-tree)

A B C D E

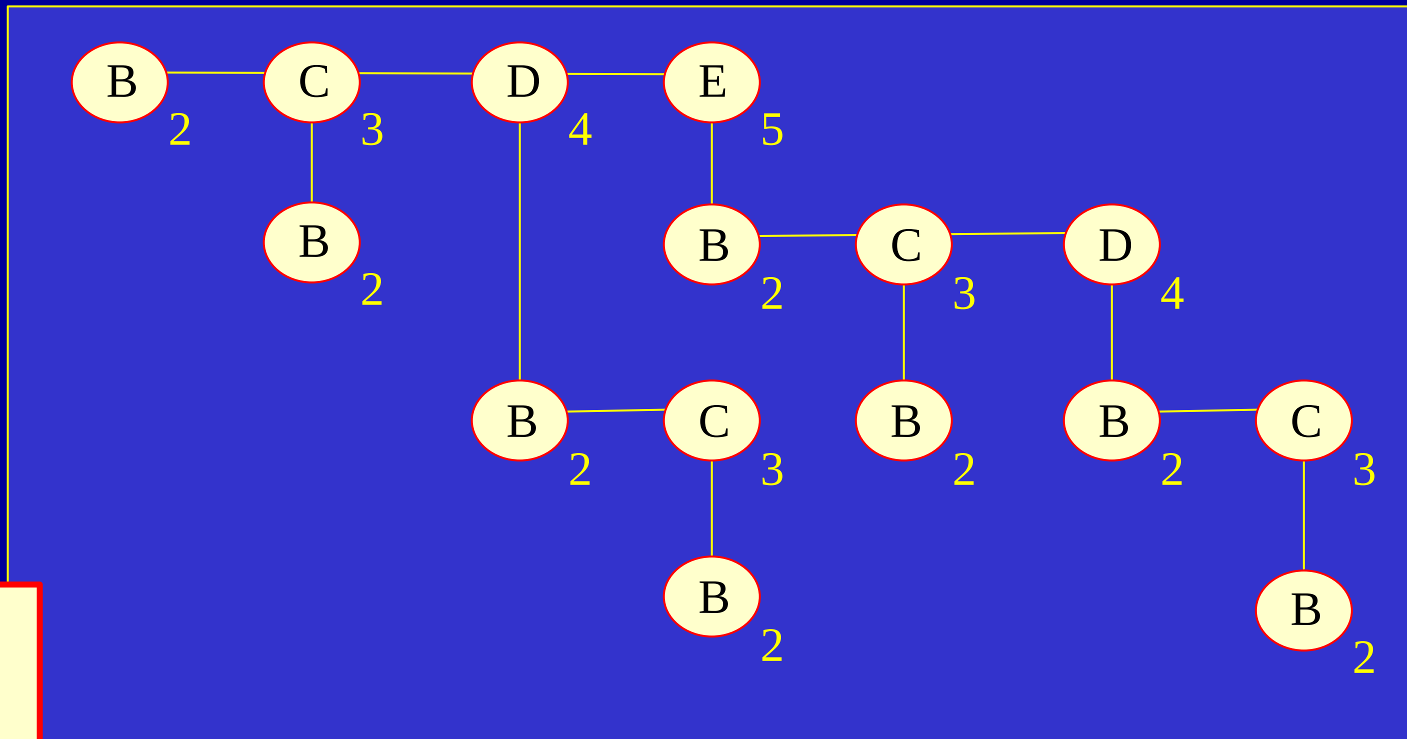
B C D E

C D E

D E

E

The Total Support Tree (T-tree)



A B C D E

B C D E

C D E

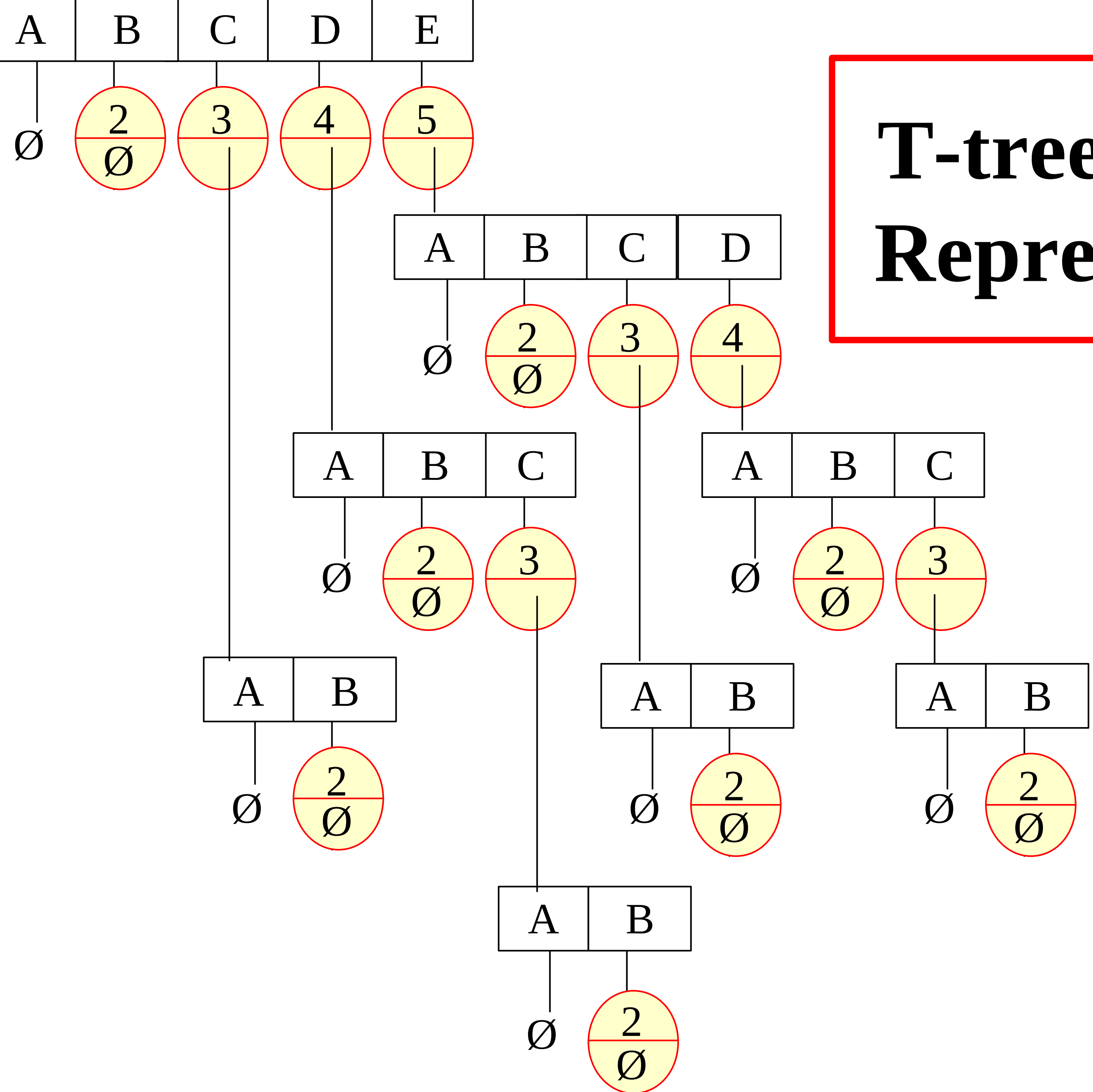
D E

E

Support Threshold = 25%

Number of frequent sets = 15

T-tree Internal Representation



The Apriori-T Algorithm

- Combines classic Apriori algorithm with T-tree data structure, i.e. tree generated level by level.
- Candidate K itemsets are produced using "*downward closure property of itemsets*".
- Includes "X-checking" --- neighbouring branches of the T-tree (sofar) inspected to determine if a given K-1 subset is supported.
- X-checking has a corresponding overhead.

Note: Authors have developed other tree based ARM algorithms, e.g. Apriori-TFP.

Advantages

1. Fast traversal of the tree using indexing mechanisms, and
2. Reduced storage, in that itemset labels are not required to be explicitly stored; thus no sibling references/pointers are required (although this is partially offset by storage required for array elements associated with roots of unsupported branches).

Distributed/Parallel ARM

- Still a desire to (i) analyse increasingly larger data sets and (ii) achieve better computational effectiveness.
- One possible solution is distributed/parallel ARM

Distributed ARM algorithms may be classified according to two categories

1. Data distribution (*segmentation* and *partitioning*).
2. Candidate set distribution (also called task distribution).

Distributed Apriori-T Algorithm with Data Distribution (DATA-DD)

Features horizontal segmentation of data

DATA-DD Algorithm

- Each process generates a level K local T-tree for its allocated segment.
- Processes share level K details so that each has a complete T-tree up to level K.
- Processes prune their versions of the T-tree sofar.
- Repeat for further levels until no more candidate sets.

Principal Disadvantage: Messaging Overhead in terms of (i) number of messages and (ii) size of content of messages.

Distributed Apriori-T Algorithm with Task Distribution (DATA-TD)

Each process has access to the entire data set and candidate sets distributed amongst processes

DATA-TD Algorithm

- Each process generates its level K candidate sets according to some agreed approach (“round robin”, partitioning).
- Process generates a level K local T-tree for their candidate sets.
- Processes share level K details so that each has a complete T-tree up to level K.
- Processes prune their versions of the tree sofar.
- Repeat for further levels until no more candidate sets.

Principal Disadvantage: Messaging Overhead in terms of (i) number of messages and (ii) size of content of messages (but less than DATA-DD which shares data about **all** candidate sets).

<u>D</u>istributed <u>A</u>priori-<u>T</u> <u>A</u>lgorithm with <u>V</u>ertical <u>P</u>artitioning (DATA-VP)

Features “vertical” partitioning of data.

DATA-VP Algorithm

- Each process generates an entire T-tree for its partition (X-checking only within partitions)
- On completion processes share T-tree details so that each has a complete global T-tree.

Vertical Partitioning

Single attributes in the data set split so that each process has its own `allocationItemSet` (AIS).

```
allocationItemSet =
    {n¦startColNum<n<=endColNum}
```

VP Algorithm

```
∀records in input data
   if (record ∩ allocationItemSet ≡ true)
      record={n¦n in record & n<=endColNum}
   else delete record in input data
```

DATA-VP Example 1

A B C D E

B C D E

C D E

D E

E

DATA-VP Example 1

AIS = {A,B,C}

A B C D E
B C D E
C D E
D E
E

AIS = {D,E}

Data Partitioning

DATA-VP Example 1

AIS = {A,B,C}

A B C

B C

C

-

-

A B C D E

B C D E

C D E

D E

E

Data Partitioning

AIS = {D,E}

DATA-VP Example 1

AIS = {A,B,C}

A B C
B C
C
-
-

A B C D E
B C D E
C D E
D E
E

Data Partitioning

AIS = {D,E}

A B C D E
B C D E
C D E
D E
E

DATA-VP Example 1

AIS = {A,B,C}

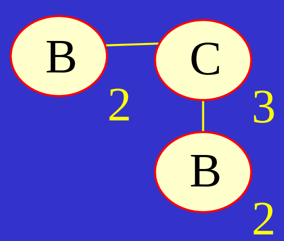
A B C
B C
C
-
-

A B C D E
B C D E
C D E
D E
E

Data Partitioning

AIS = {D,E}

A B C D E
B C D E
C D E
D E
E



Num. frequent sets = 3

Support Threshold = 25%

DATA-VP Example 1

AIS = {A,B,C}

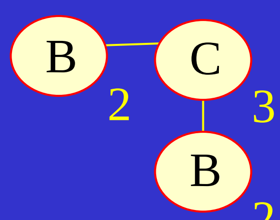
A B C D E
B C D E
C D E
D E
E

Data Partitioning

AIS = {D,E}

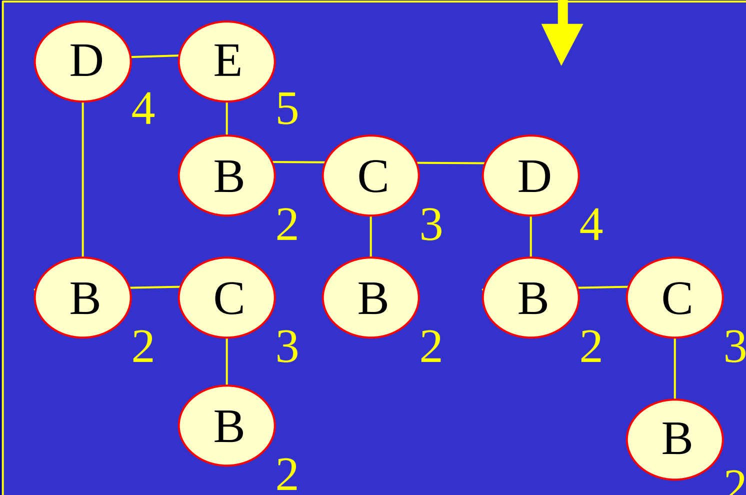
A B C
B C
C
-
-

A B C D E
B C D E
C D E
D E
E



Num. frequent sets = 3

Support Threshold = 25%



Num. frequent sets = 12

DATA-VP Example 2 (With Data Reordering)

A B C D E
B C D E
C D E
D E
E

DATA-VP Example 2 (With Data Reordering)

E D C B A
E D C B
E D C
E D
E

Data Reordering

A B C D E
B C D E
C D E
D E
E

DATA-VP Example 2 (With Data Reordering)

AIS = {E,D,C}

E D C B A
E D C B
E D C
E D
E

AIS = {B,A}

Data Reordering

A B C D E
B C D E
C D E
D E
E

DATA-VP Example 2 (With Data Reordering)

AIS = {E,D,C}

E D C
E D C
E D C
E D
E

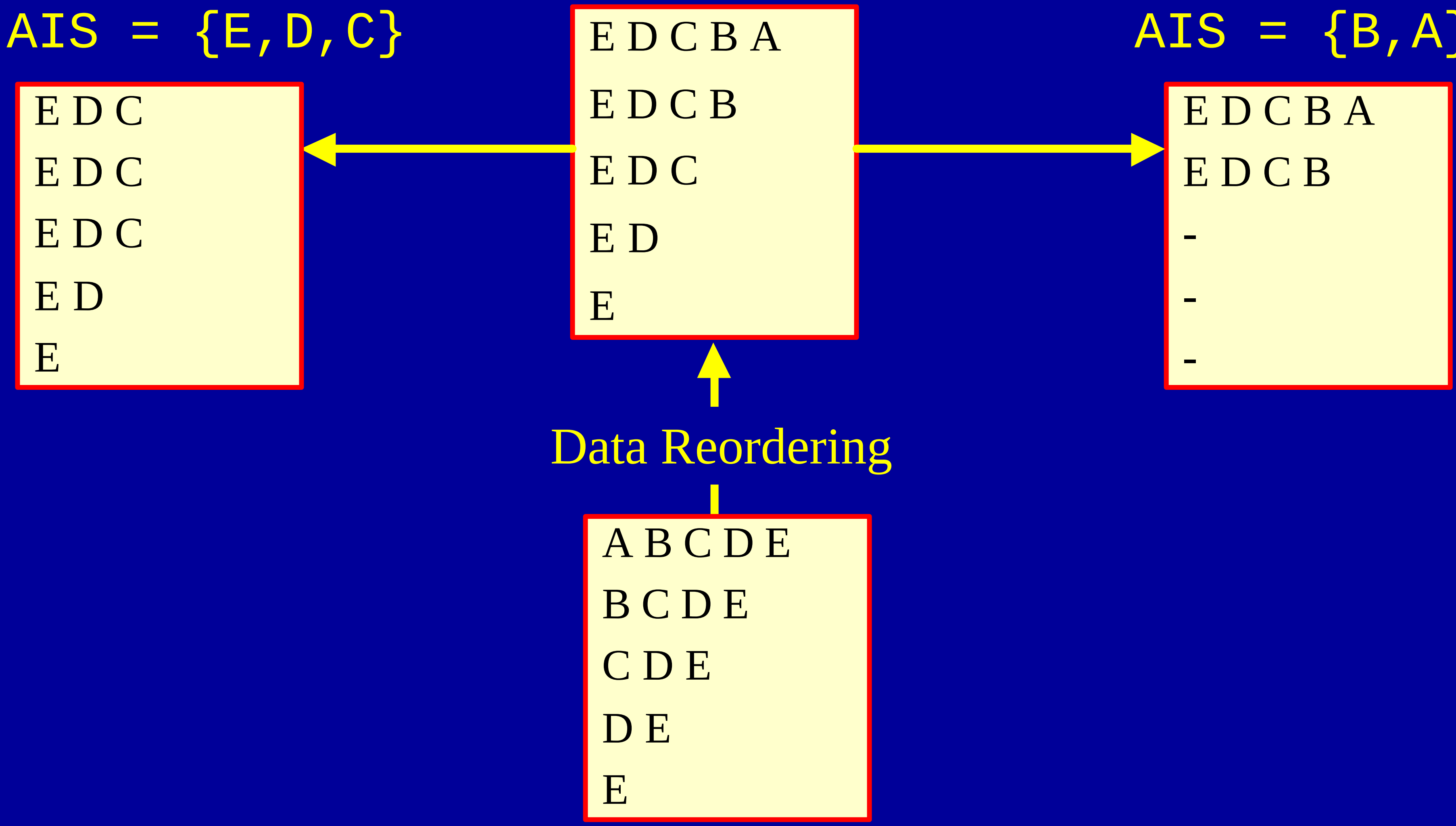
E D C B A
E D C B
E D C
E D
E

AIS = {B,A}

Data Reordering

A B C D E
B C D E
C D E
D E
E

DATA-VP Example 2 (With Data Reordering)



DATA-VP Example 2 (With Data Reordering)

Support Threshold = 25%

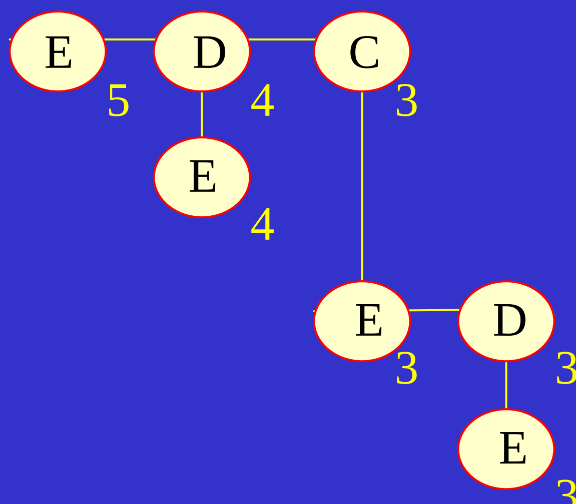
A B C D E
B C D E
C D E
D E
E

Data Reordering

E D C B A
E D C B
E D C
E D
E

AIS = {E,D,C}

E D C
E D C
E D C
E D
E



Num. frequent sets = 7

AIS = {B,A}

E D C B A
E D C B
-
-
-

DATA-VP Example 2 (With Data Reordering)

AIS = {E,D,C}

E D C
E D C
E D C
E D
E

E D C B A
E D C B
E D C
E D
E

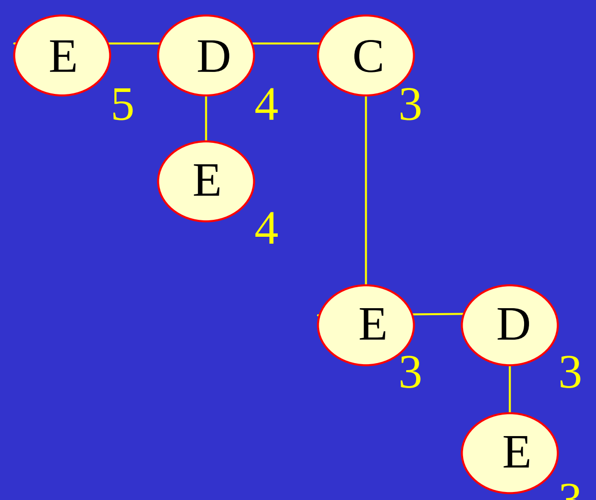
AIS = {B,A

E D C B A
E D C B
-
-
-

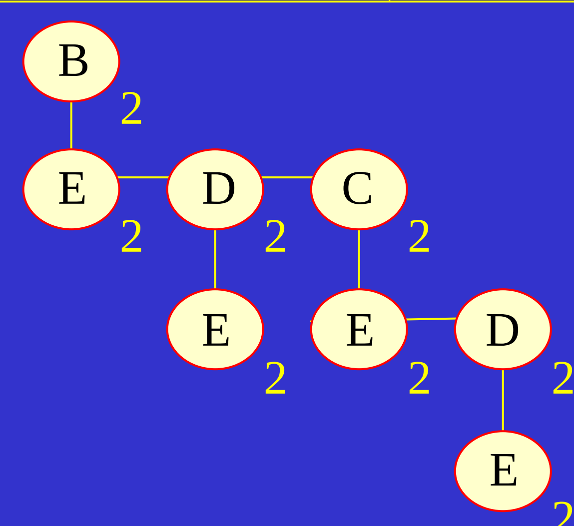
Data Reordering

A B C D E
B C D E
C D E
D E
E

Support Threshold = 25%



Num. frequent sets = 7



Num. frequent sets = 8

Some Results (T20.I10.D500K.N500)

Processing time (Seconds)

ALGORITHM	# Processes	Support %			
		2.0	1.5	1.0	0.5
Apriori-T	1	15	19	31	95
DATA-DD	5	13	16	25	99
DATA-TD	5	9	9	16	66
DATA-VP	5	3	4	10	31

Implementation: Java, JavaSpaces

Advantages of DATA-VP

- Minimal amount of message passing compared to DATA-DD and DATA-TD.
- Minimal message size, especially with respect to DATA-DD.
- Enhanced efficiency as the number of processes increases, unlike DATA-DD.

Summary and Conclusions

1. Experiments show that the DATA-VP approach performs much better than those methods that use data and task distribution (especially if we order the data).
2. This is largely due to the T-tree data structure which: (a) facilitates vertical partitioning of the input data, and (b) readily lends itself to distribution/parallelisation.
3. More generally we have demonstrated that both the T-tree data structure and the Apriori-T algorithm are good generic mechanisms that can be used effectively to implement many approaches to distributed/parallel ARM.