Parallel Data Mining

Alexandre Termier
LIG laboratory, HADAS team
`Alexandre.Termier@imag.fr`

Data Mining

- What ?

  Apply computational techniques to "*identify valid, novel, potentially useful, and ultimately understandable patterns in data*" [Fayyad, 96]

- Why ?
  - Large quantity of data
  - Human analysis doesn't scale up

Major domains of Data Mining

- Classification
  - Classify data according to a know set of classes
  - Simple example : classify mails in your mailbox
  - More complex example : classify Reuteurs news according to the classes {Politics, Economy, Science, Sports}

- Clustering
  - Discover (and possibly characterize) « custers » in data
    - Intra-cluster similarity must be high
    - Inter-cluster similarity must be low
  - Example : Discover several groups of patients in medical experiments

- Frequent pattern mining
  - Discover patterns in data whose frequency is more than a given threshold
  - More examples to come…

Outline

1. Introduction to Data Mining

3. Frequent pattern mining algorithms (sequential)

5. Parallel pattern mining algorithms

The supermarket problem

Transactions

T_1

- Bread
- Butter
- Chocolate

T_2

- Bread
- Orange juice
- Chocolate
- Pencil

T_3

- Bread
- Cheese
- Vine

Items

66% of transactions contain Bread + Chocolate (T_1, T_2)

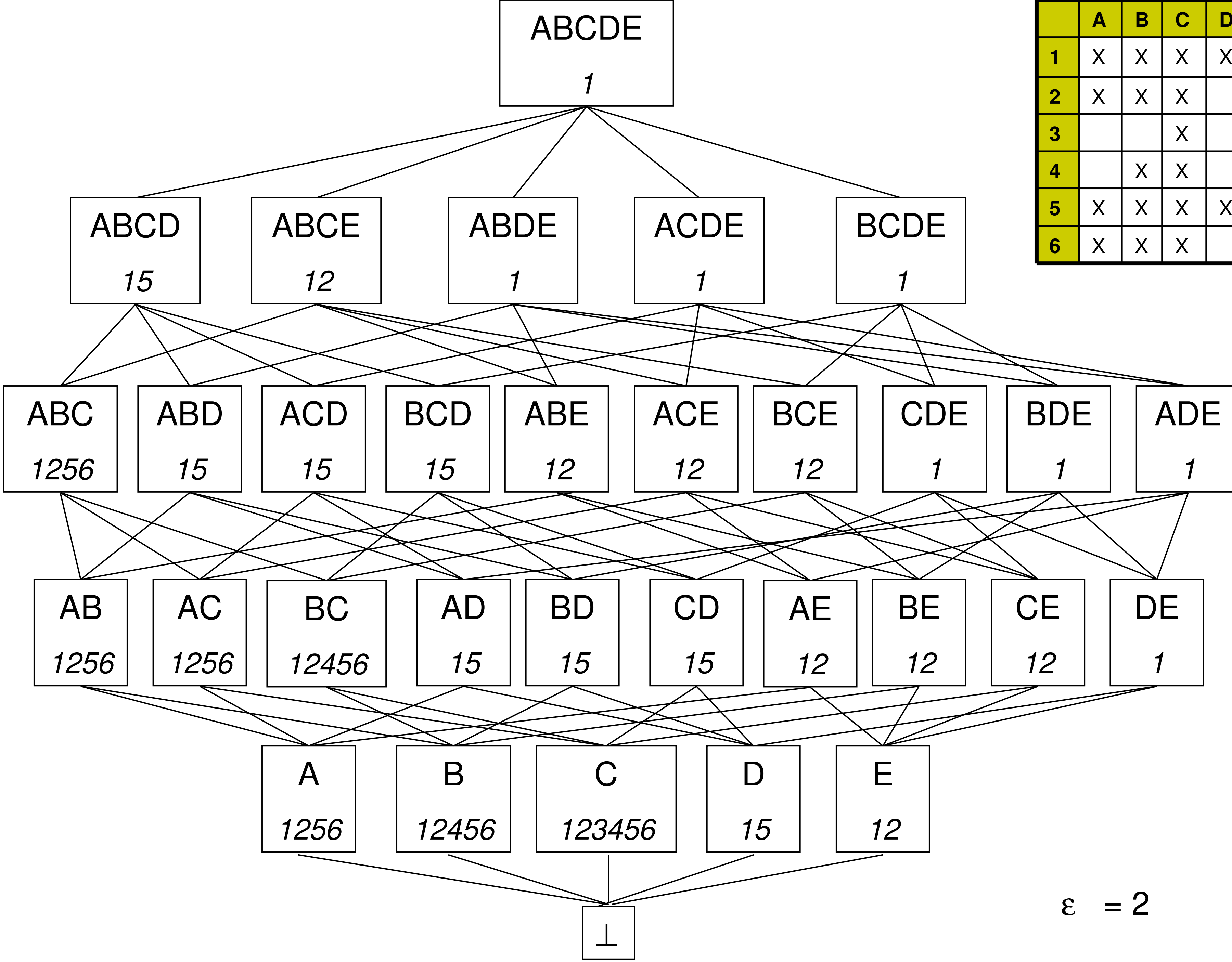
Support **(frequent) itemset** **Tid-list**

How to compute frequent itemsets ?

- Generate and Test
  - Generate a candidate itemset
  - Test its frequency against the database
  - Highly combinatorial problem !
    - 1000 items $\rightarrow 2^{1000}$ possible itemsets

- Apriori algorithm [Agrawal *et al.*, 93]
  - Levelwise generation
    - Generate candidate of depth 1, then 2, then 3…
  - Anti-monotonicity pruning
    - *All super-itemsets of an infrequent itemset are also infrequent*

	A	B	C	D	E
1	X	X	X	X	X
2	X	X	X		X
3			X		
4		X	X		
5	X	X	X	X	
6	X	X	X		



$\varepsilon = 2$

Apriori performance

- Complexity
  - Scales linearly with #transactions
  - Scales exponentially with #items
- Usable on datasets having few frequent itemsets of small size
- Lots of research for improving this algorithm
  - Sampling database [Toivonen *et al*., 96]
  - Partition [Brin *et al*., 97]
  - FP-Growth [Han *et al*., 00]

Closed itemsets (Pasquier *et al.,99*)

- **Definition**
  - F closed if all F' $\supset$ F have a strictly smaller tid-list
- **Property**
  - If F closed and F' $\subset$ F, F' not closed, then:

    tid-list(F) = tid-list(F')
  - All FIS can be constructed from the set of closed FIS
- **Gain**
  - Exponential gain (in size of FIS)
- **Algorithm**
  - LCM2 algorithm [Uno *et al*., 2004]
    - enumerates the tree of closed frequent itemsets by *ppc-extension*

Closed frequent itemsets

Transactions:

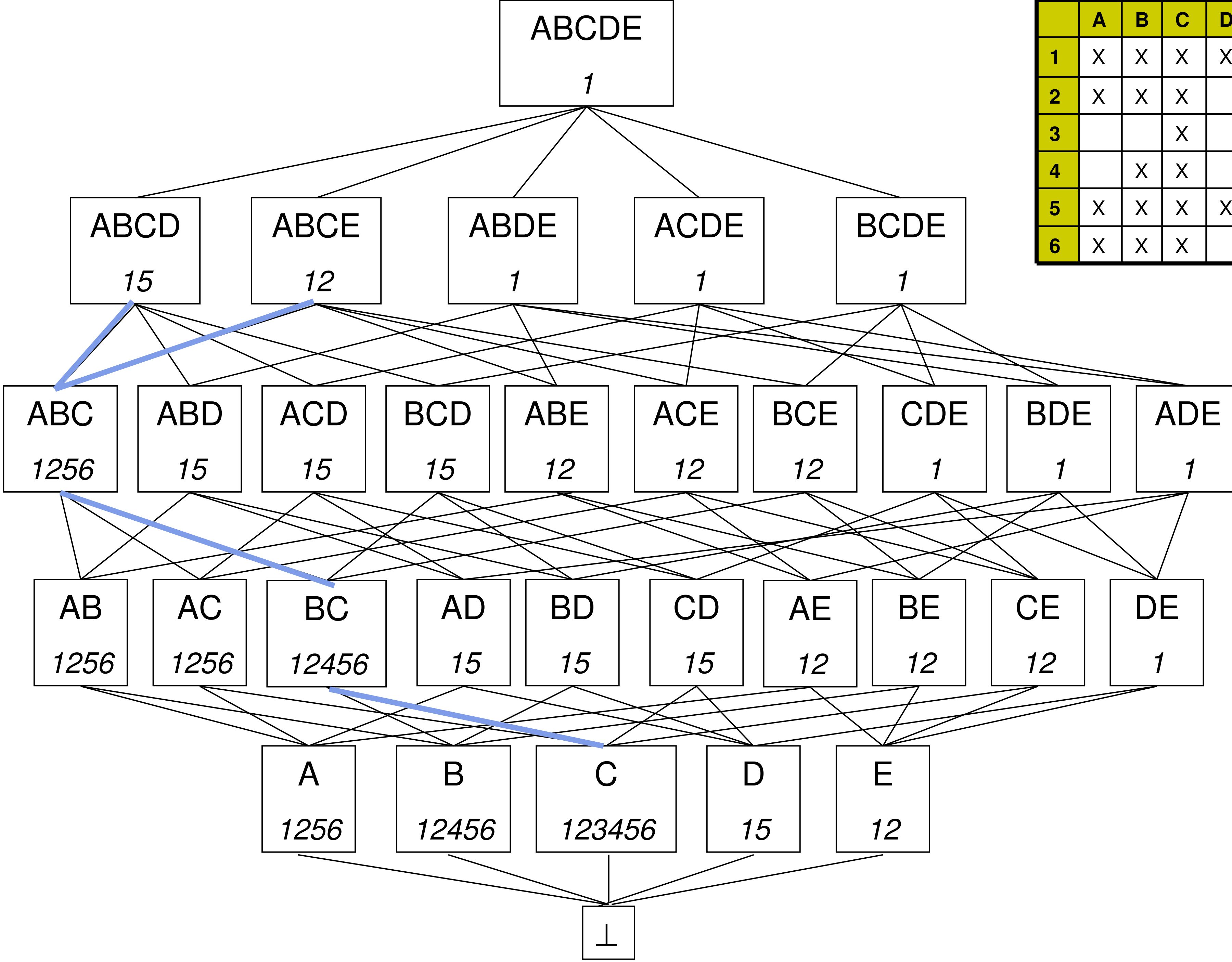
items

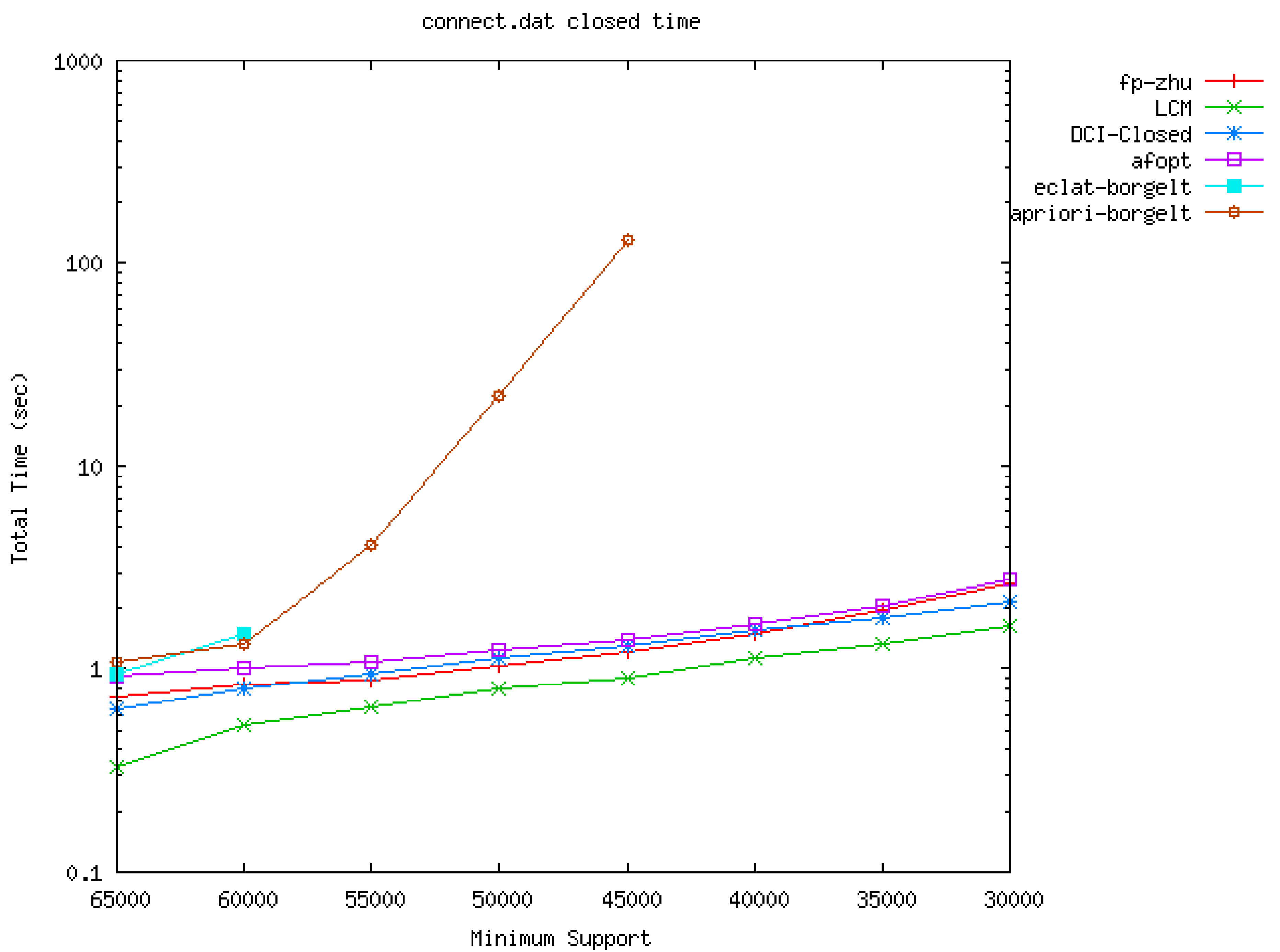
T1	T2	T3
• Bread • Butter • Chocolate	• Bread • Orange juice • Chocolate • Pencil	• Bread • Cheese • Vine

$\mathcal{E} = 66\%$

	tidlist		support
closed frequent itemset	{T1, T2, T3}	contains {Bread}	*(100%)*
frequent itemset	{T1, T2}	contains {Chocolate}	*(66%)*
closed frequent itemset	{T1, T2}	contains {Bread, Chocolate}	*(66%)*

	A	B	C	D	E
1	X	X	X	X	X
2	X	X	X		X
3			X		
4		X	X		
5	X	X	X	X	
6	X	X	X		

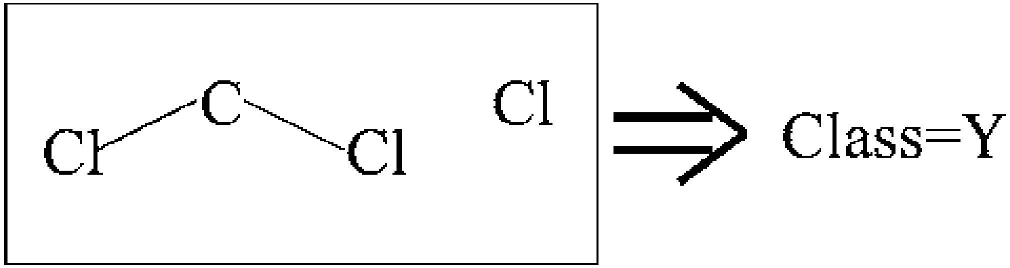



connect.dat closed time
Total Time (sec)
1000
100
10
1
0.1
65000
60000
55000
50000
45000
40000
35000
30000
Minimum Support
fp-zhu
LCM
DCI-Closed
afopt
eclat-borgelt
apriori-borgelt

Mining structured patterns

- Beyond itemsets : structured patterns
  - Sequences (marketing)
  - Trees (phylogenetic trees, web logs)
  - Graphs (bioinformatics, chemistry)

- Chemistry example (from [Inokuchi *et al*., 02])

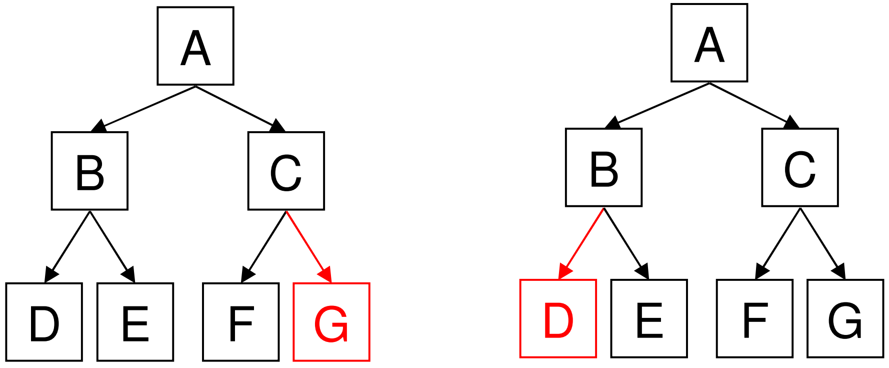


Support : 31,7%

Dataset : 41 organic chlorydes, 31 of which are carcinogenic

Difficulties

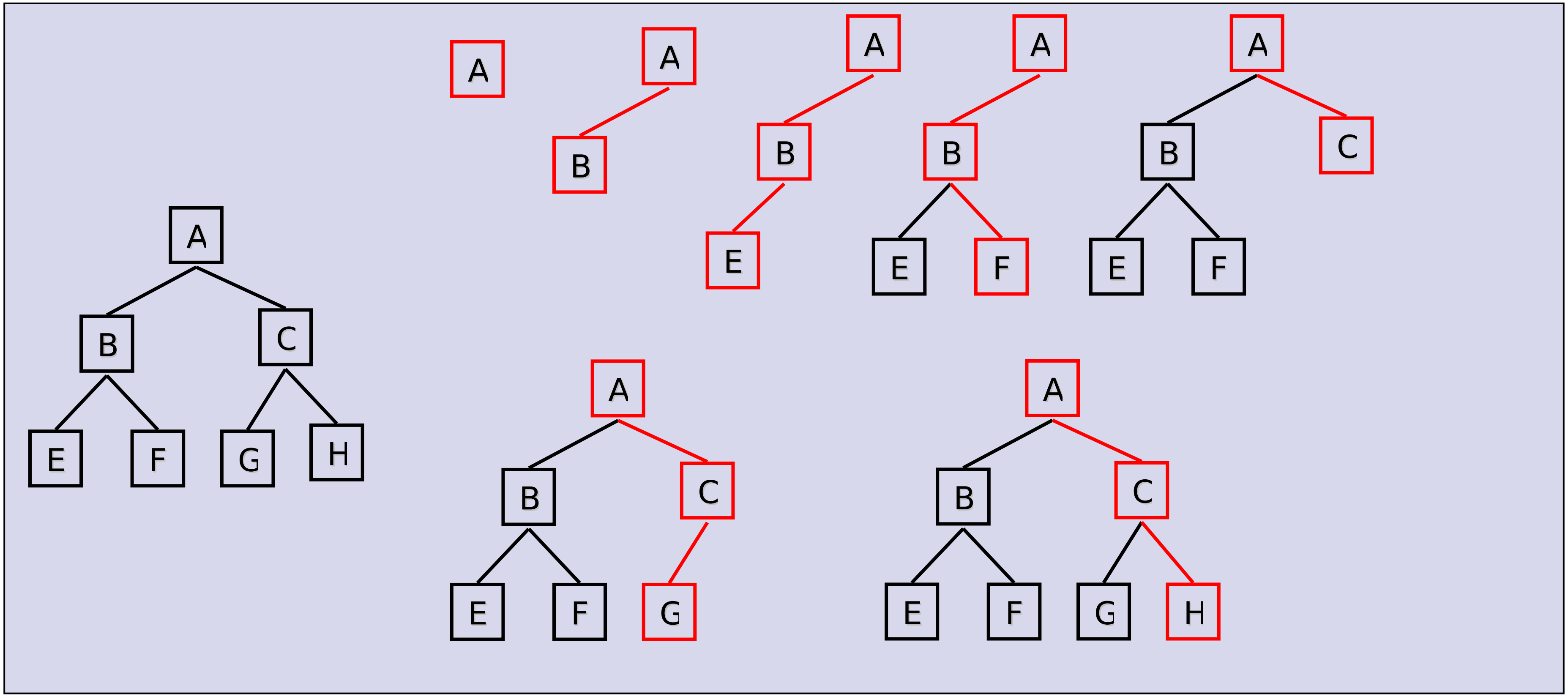
- Combinatorial much higher than itemsets → huge search space
- Possibility to generate twice the same candidate
  - need specific enumeration techniques



- Frequency test = {sequence/tree/graph} inclusion
  - For trees, more than 10 different inclusions exists [Kilpelainen, 92]
  - The most interesting are NP-complete to check

Tree Mining

- Freqt [Asai *et al*., 02] and TreeMiner [Zaki, 02]
  - Rightmost branch expansion

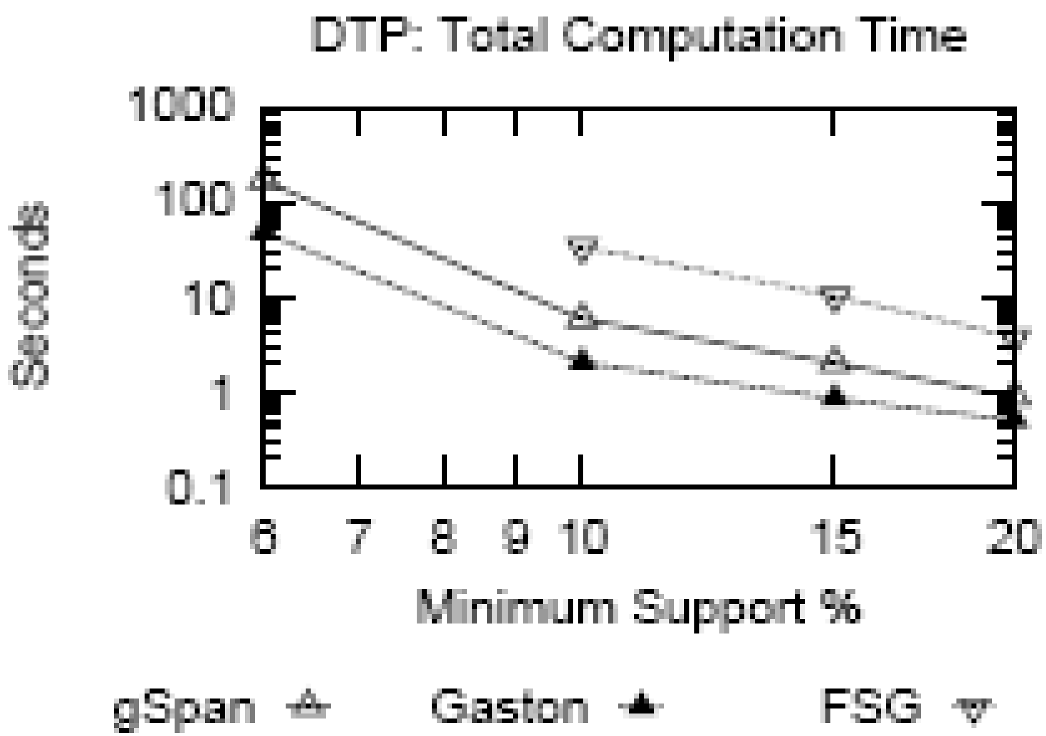


Graph Mining

- **A**priori **G**raph **M**iner [Inokuchi *et al*., 00]
  - Same principles as Apriori : levelwise, antimonotonicity
  - Graphs are represented by matrices
- gSpan : FP-growth for graphs
- Gaston : find pathes → free trees → graphs

Run times

- Dataset: 422 molecules
  - Average 40 vertices, 42 edges (max : 188 vertices, 196 edges).
  - 21 kinds of different atoms
- Machine: Athlon XP1600+, 512 MB RAM



Parallel Pattern Mining

- Pattern Mining : huge need of performance
- → in ~5 years, pattern mining = parallel pattern mining

- Problems :
  - How to design a parallel pattern mining algorithm ?
  - Implementation ?
  - Scalability ?

Distributed frequent itemsets miners

- Years 1996~2001: target architecture = clusters
- Main problem: **load unbalance**
- Count Distribution [Agrawal *et al*, 96]
  - Each node has a partial database
  - Locally counts support
  - At the end of each iteration, merge local supports to compute global supports
  - → Lots of communications / synchronization
- Candidate Distribution [Agrawal *et al*, 96]
  - Selective replication of database
  - Processors work independantly on local portions of database
  - Better performance
- Improvement by [Zaki *et al*, 97]
  - Use classes of equivalence for candidate partitioning

"Mining on emergent architectures"

- Parallel gSpan [Buehrer *et al*, 06]
  - Adaptative mining (based on depth first)
  - Work queuing / work stealing
  - Pack some child tasks together : improve temporal locality

- Multicore → strong Intel support

- **Ad-hoc parallelisation of existing algorithms**

Our goal at Hadas/Mescal

- Provide a multi-platform, multi-algorithm framework for parallel pattern mining
  - Multicore / Clusters
  - Itemset / Trees / DAGs / Graphs …

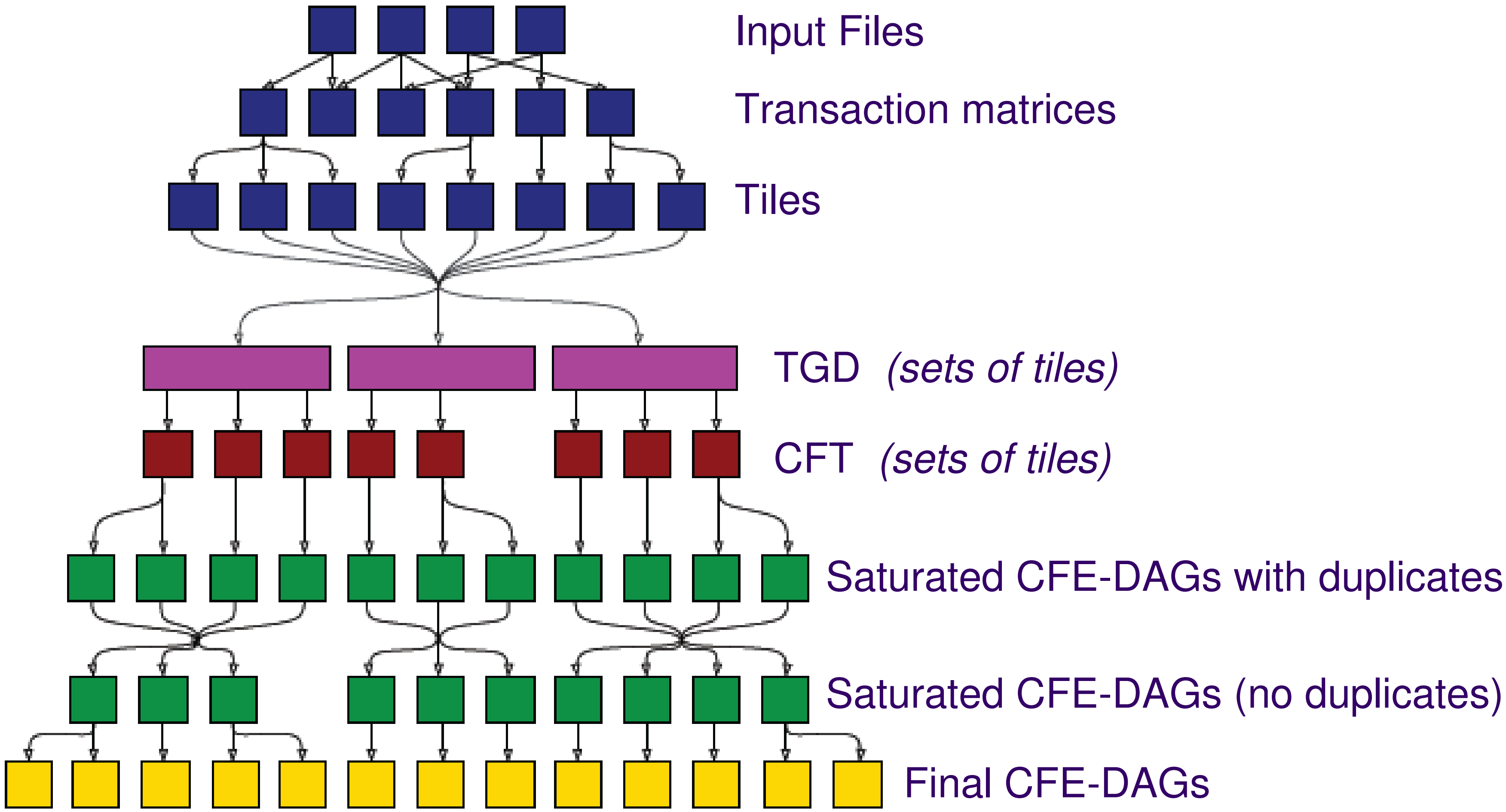
- Easy to use by DM researchers / engineers
  - No explicit parallel instructions in algorithm

- Good performance
  - Scalability up to hundreds of cores

Target algorithm : DigDag

- DigDag mines *closed frequent embedded DAGs (CFE-DAGs)*
- Application : gene networks (bioinformatics)
- Sequential version need hours to complete with low support thresholds

DigDag flow


Input Files
Transaction matrices
Tiles
TGD *(sets of tiles)*
CFT *(sets of tiles)*
Saturated CFE-DAGs with duplicates
Saturated CFE-DAGs (no duplicates)
Final CFE-DAGs

Preliminary works

- Existing parallel programming environments:
  - Posix threads *(low level)*
  - Intel TBB *(task based, good for recursive algorithms)*
  - OpenMP *(simple, good for loops)*
  - MPI *(for clusters)*

- First parallel version of DigDag : DigDagOpenMP

DigDagOpenMP

- OpenMP : loop parallelization

```
vector<Object*> listObjects ;
…
int nbObjects = listObjects.size(), i ;
#pragma omp parallel for schedule(dynamic) default(shared) private(i)
for (i=0 ; i<nbObjects ; i++) {
    process(listObjects[i]) ;
}
```

DigDagOpenMP

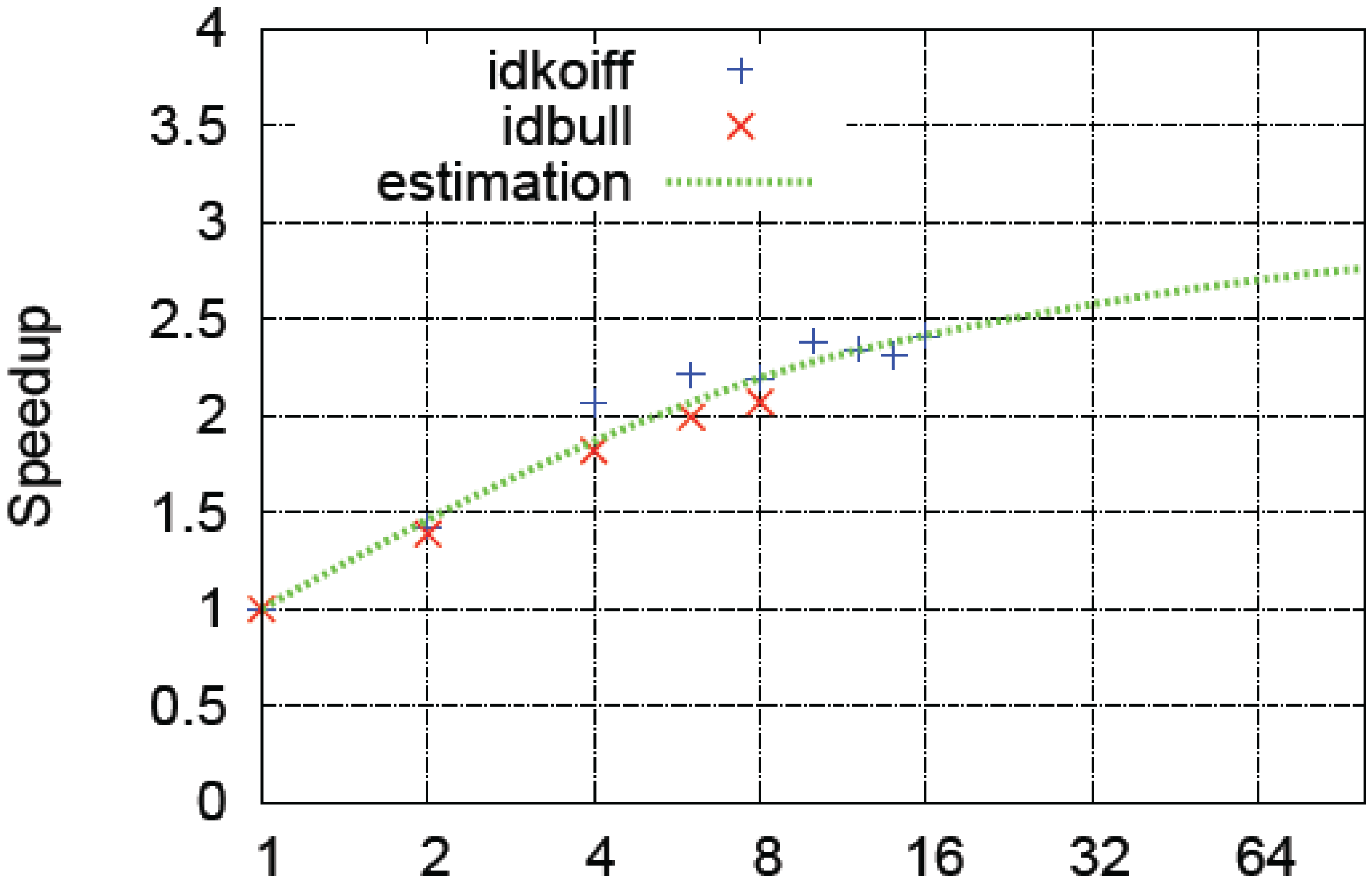
- OpenMP : critical sections

```
vector<Object*> listObjects ;
vector<Result*> listResults ;


…
int nbObjects = listObjects.size(), i ;
#pragma omp parallel for schedule(dynamic) default(shared) private(i)
for (i=0 ; i<nbObjects ; i++) {
    Result* res = process(listObjects[i]) ;
    #pragma omp critical {
        listResults.push_back(res) ;
    }
}
```

DigDagOpenMP

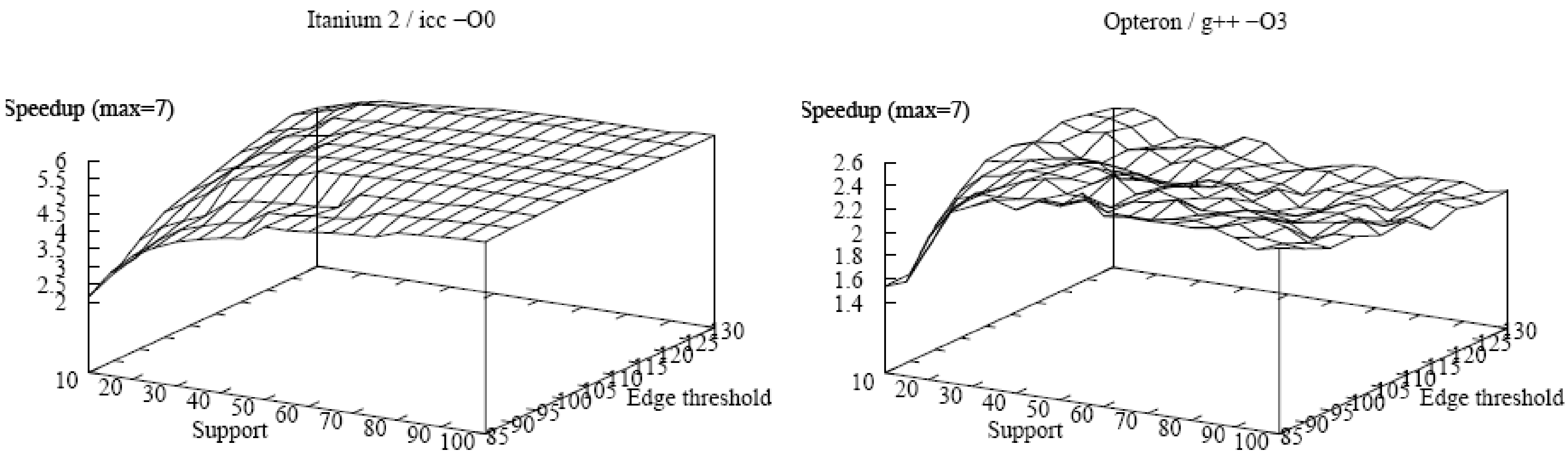
- Easy of parallelizing a sequential program
  - 1 day of work for the 6000 code lines of DigDag
  - More than 90% of code can be executed in parallel

- Experiments
  - Real data: Hugues 2000 / 300 genes / 1000 DAGs
  - 16-way Opteron 875 @ 2.2 GHz / 32 GB RAM
  - 8-way Itanium



#processors (log scale)

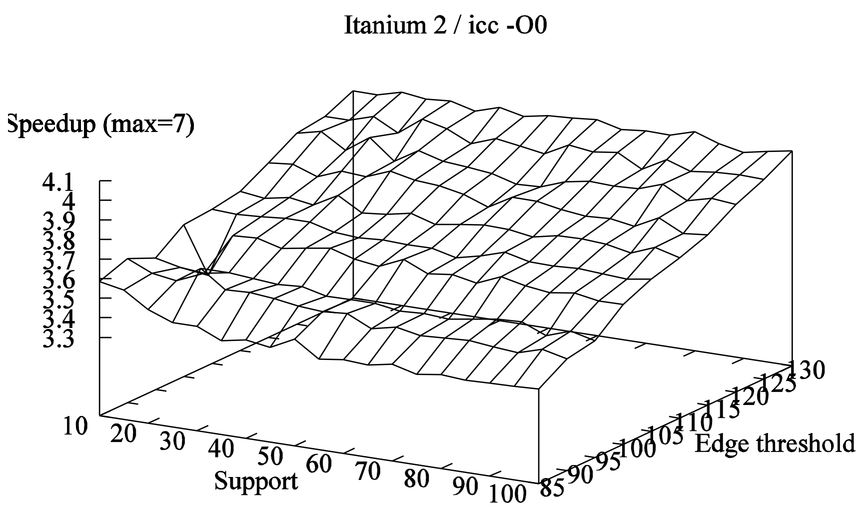
Experiments



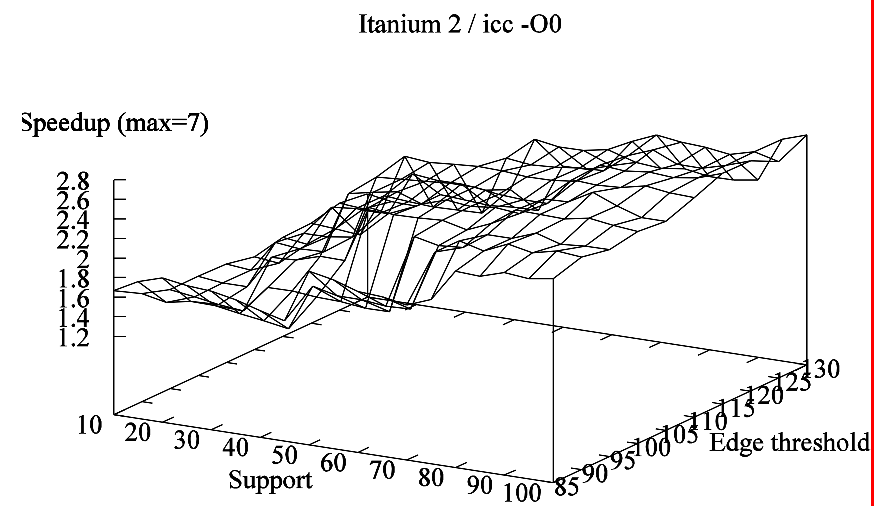
- Speed-up values deceiving, depend on :
  - Computer
  - Compiler
  - Level/Quality of compiler optimizations

- Different behavior of OpenMP implementations depending on compilers

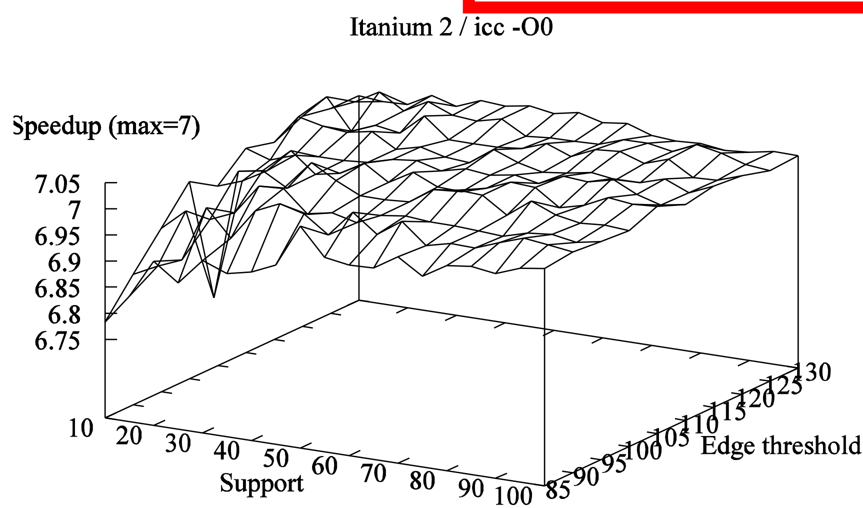
Speed-up by code part



1. Data loading and initializations

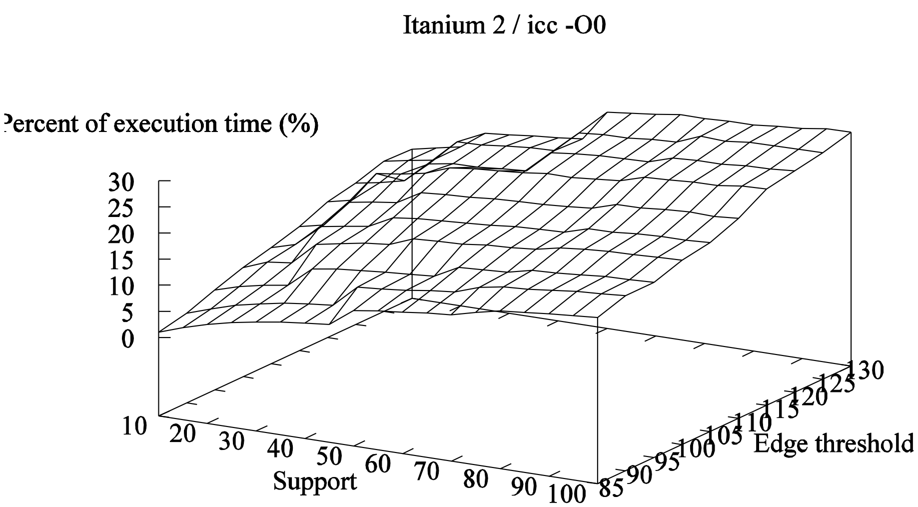


2. Computations step 1 (Tiles – TGD - CFT)

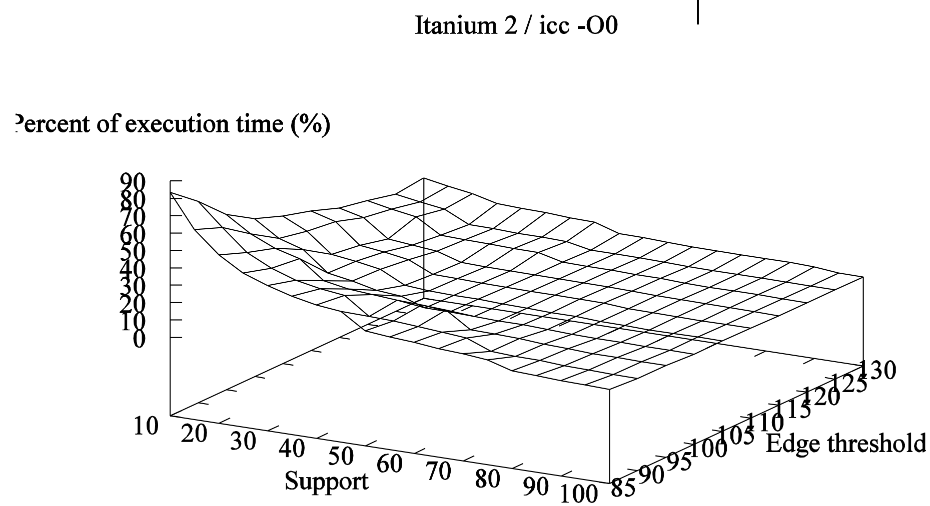


3. Computations step 2

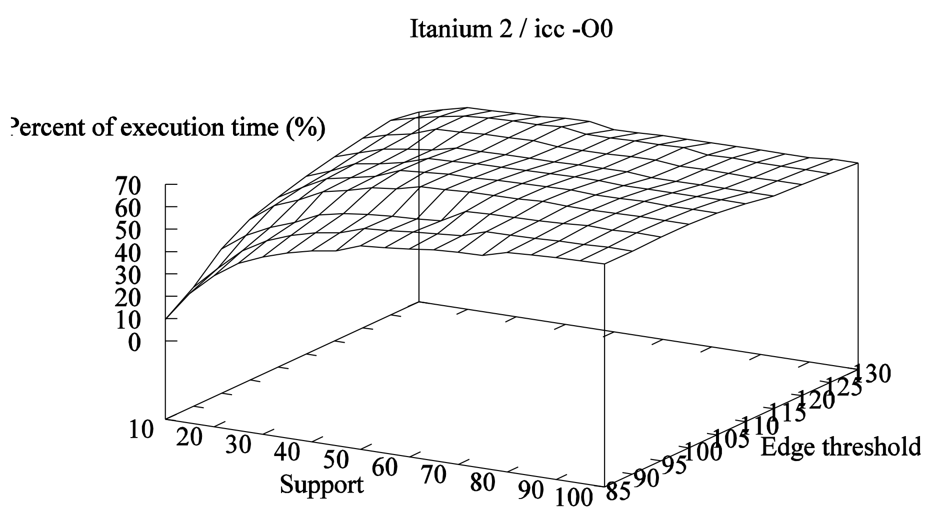
Execution time percentages



1. Data loading and initializations

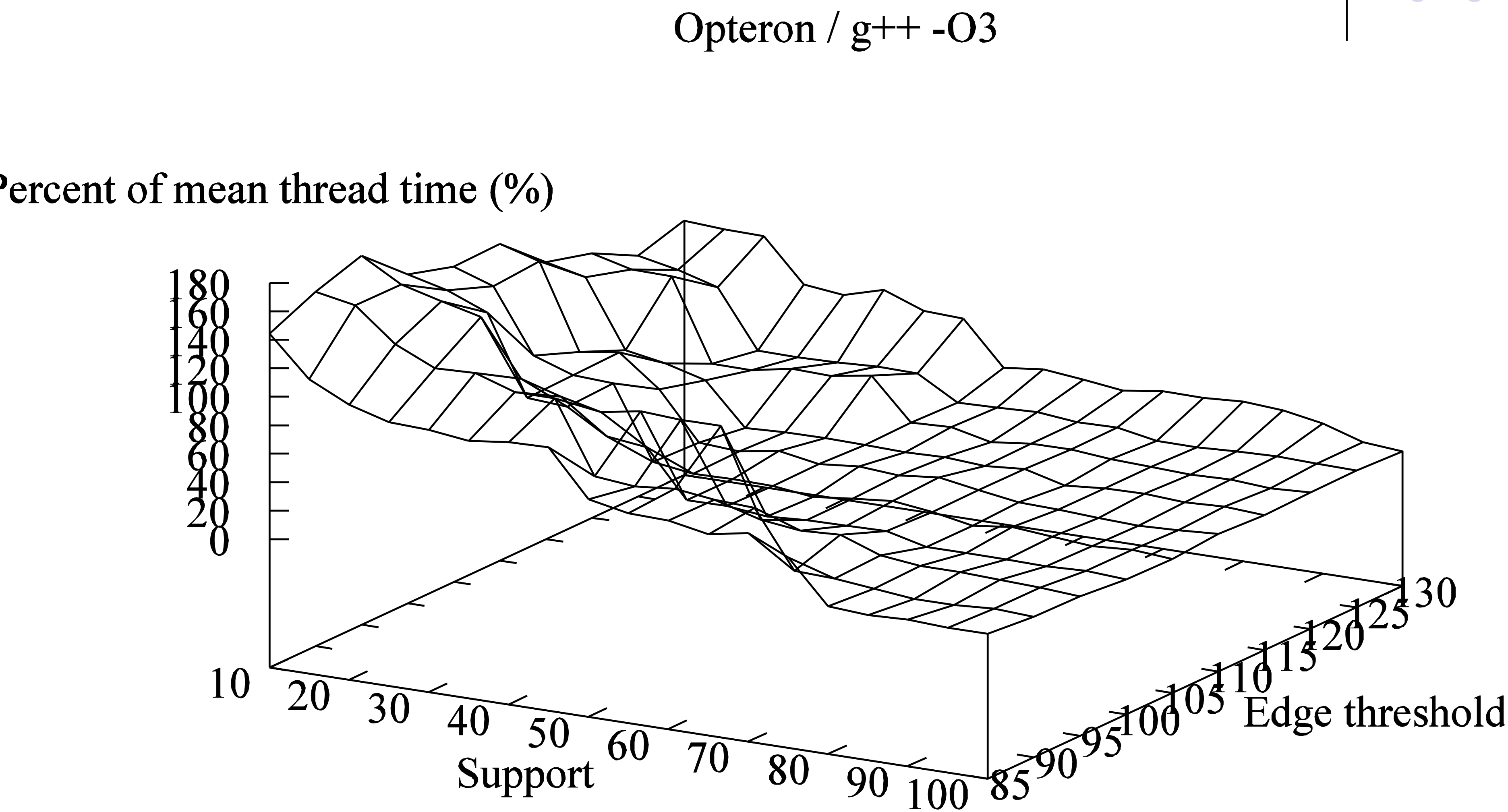


2. Computations step 1 (Tiles – TGD - CFT)



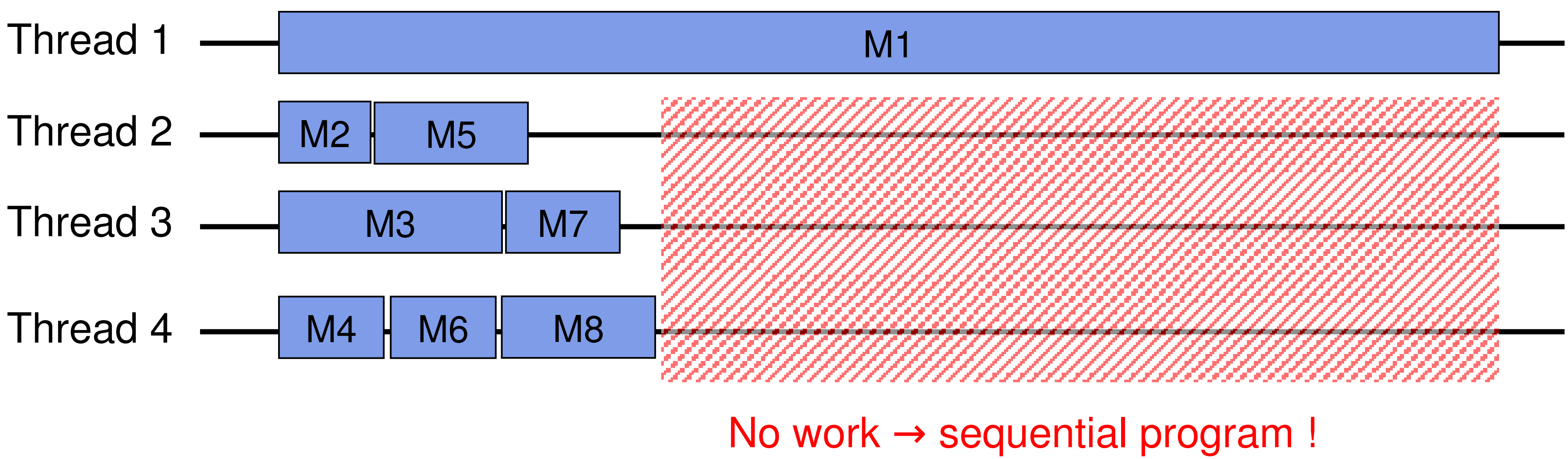
3. Computations step 2

Load unbalance



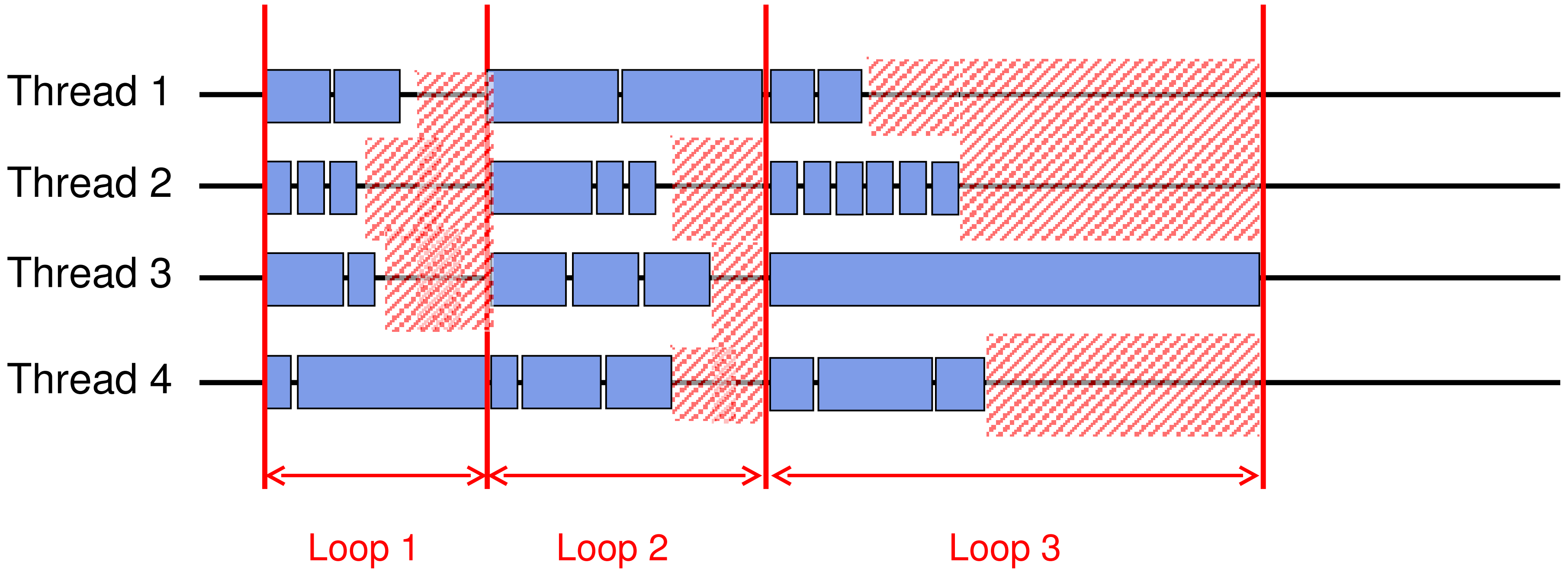
Why load unbalance ?

- Basic program operation : computing closed frequent itemsets from matrices with LCM2 [Uno *et al.*, 04]
- LCM2 : sequential
- These matrices have very different « difficulty »

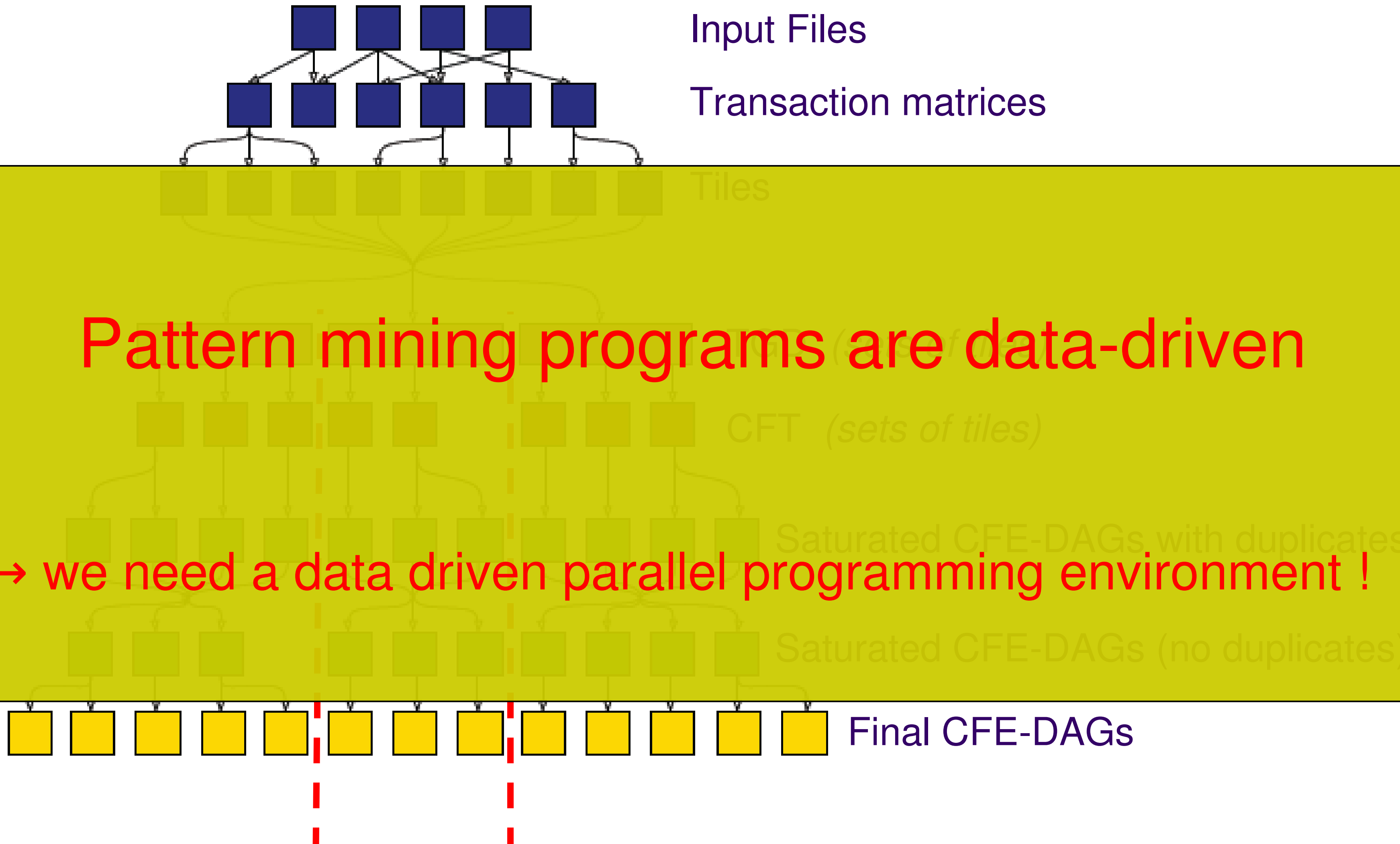


Barriers in the code

- **Barrier :** Synchronization point that all threads must reach before continuing execution



How to improve ?

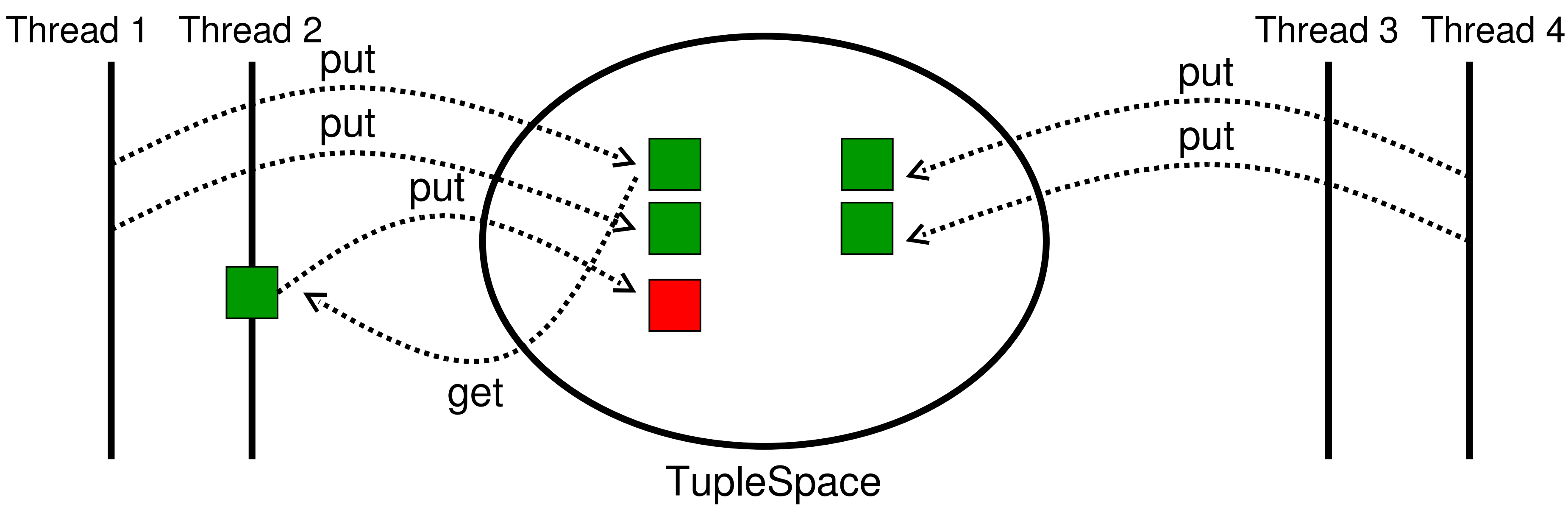


Pattern mining programs are data-driven

→ we need a data driven parallel programming environment !

Linda (Gelernter, 89)

- **Linda :** parallel programming environment centered on data.
  - Interaction between processes : add/remove tuples in a TupleSpace



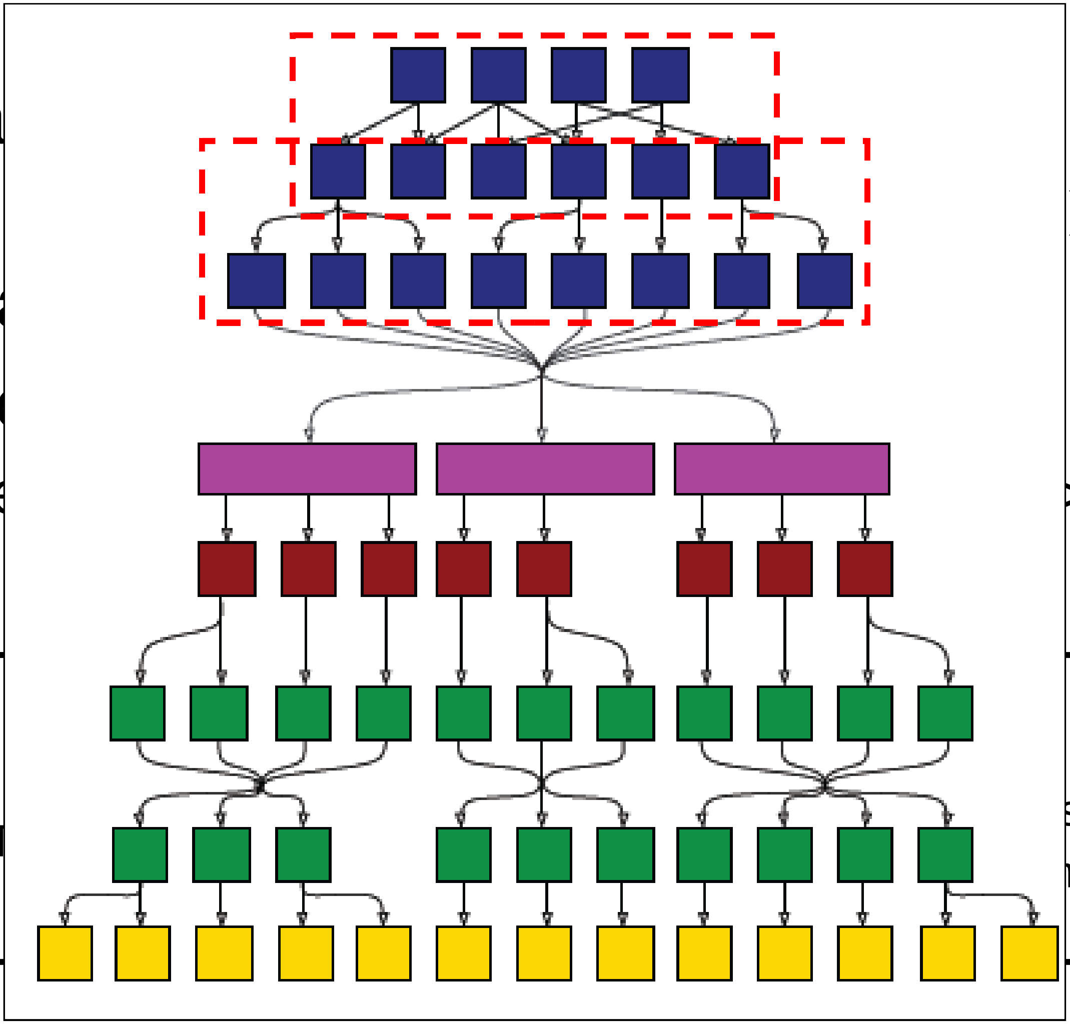
Our contribution : Melinda

- Builds up upon the idea of TupleSpaces

- Adaptations for pattern mining algorithms
  - Simplified model → can be used as a library
  - Multiple TupleSpaces
  - Typed TupleSpaces

Melinda + DigDag 1/2

- Melinda
)
- Major d eSpaces
- Simple
  - Ex : ve >

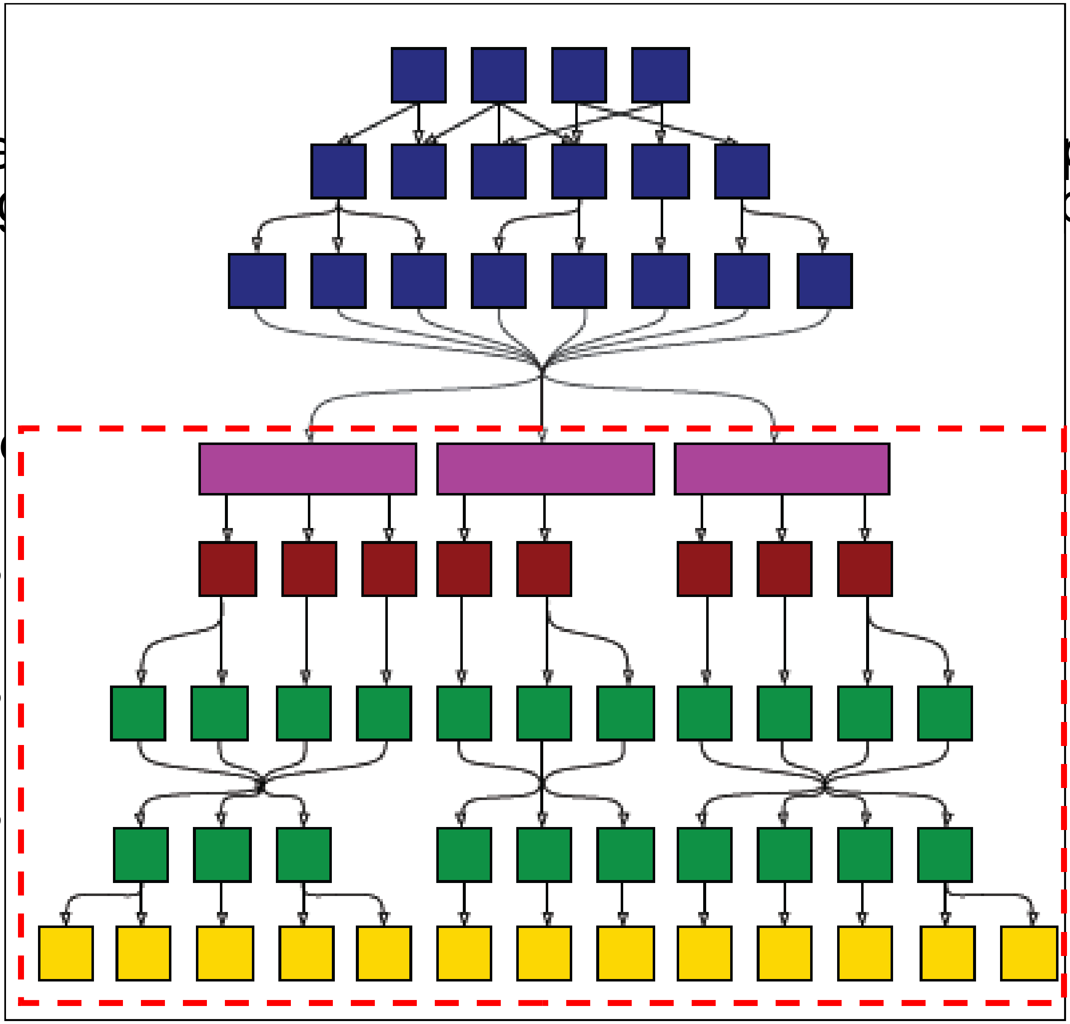


```
for (i = 0;                      s.get(&matrix))
    computeT                     matrix);
```

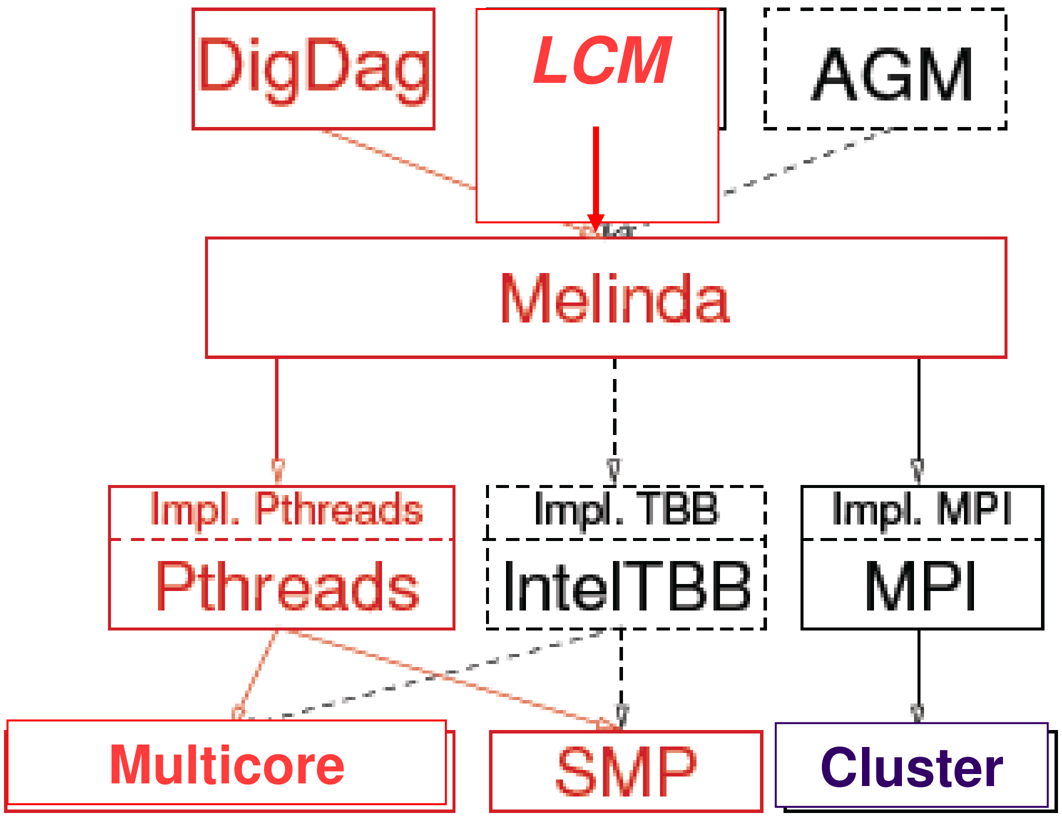
Melinda + DigDag 2/2

- When se… parallel (CFT
  – SatDag… pace

```
Data data
while(spa
   switch(
      cas

      cas

      cas

   }
}
```



Multi-platform / Multi-algorithm environment

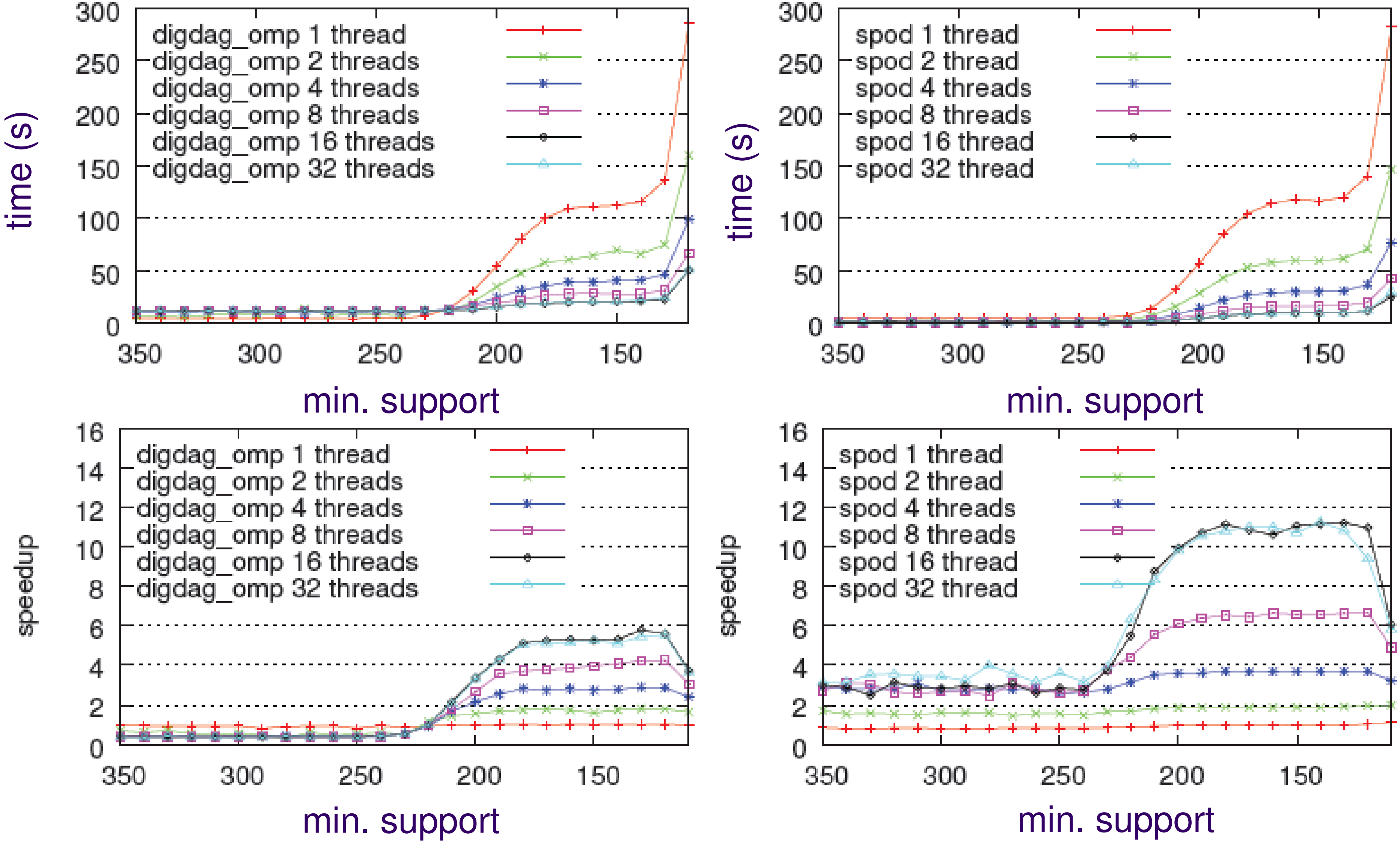


Experimental settings

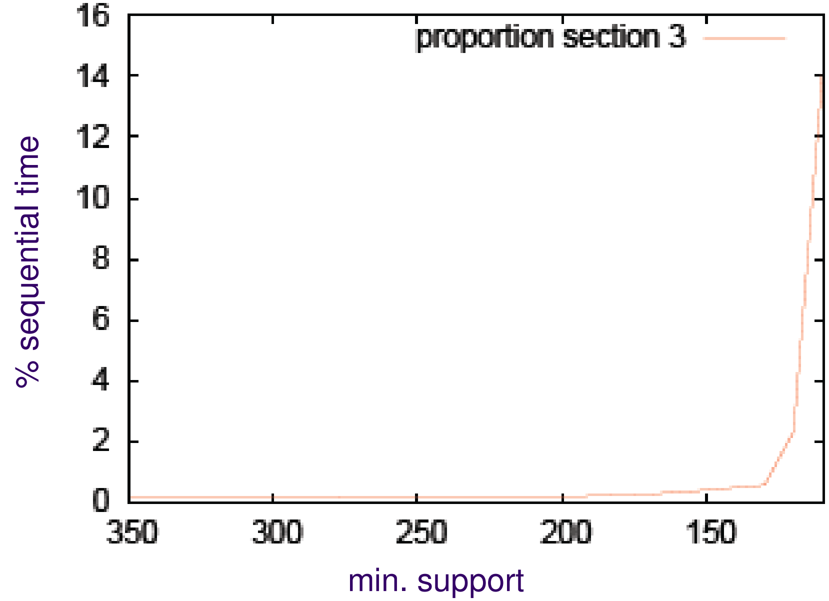
- Synthetic data
  - Random DAG generator (Bayesian Networks)
  - 600 DAGs
  - 60 nodes/DAG
  - Average branch factor : 2.5

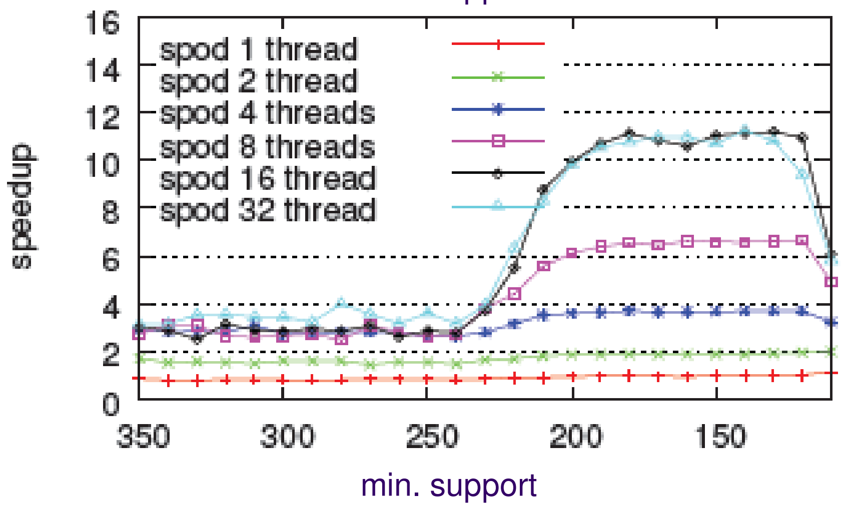
- Real data
  - Gene networks from Spellman dataset
  - 5000 DAGs
  - 801 nodes/DAG

Synthetic data

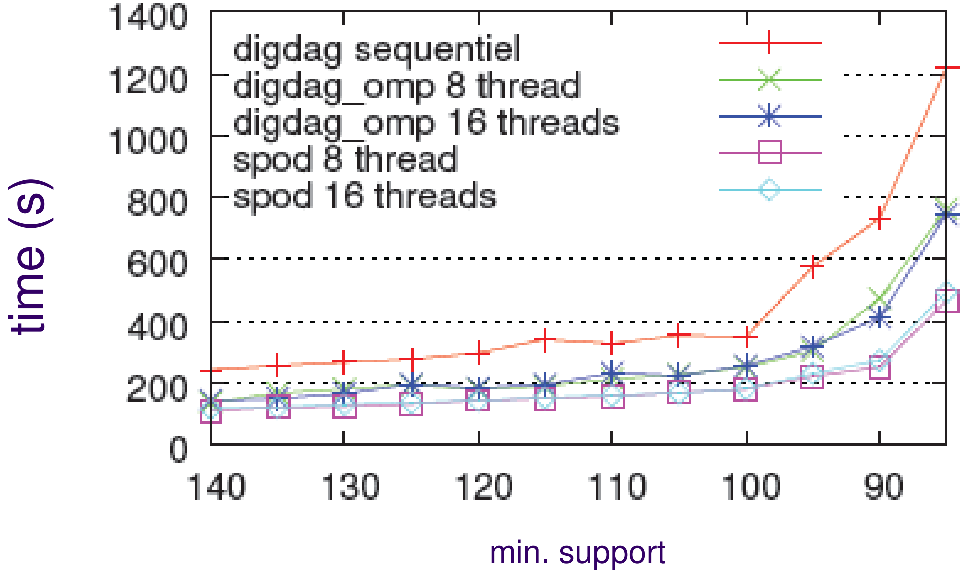


Synthetic data

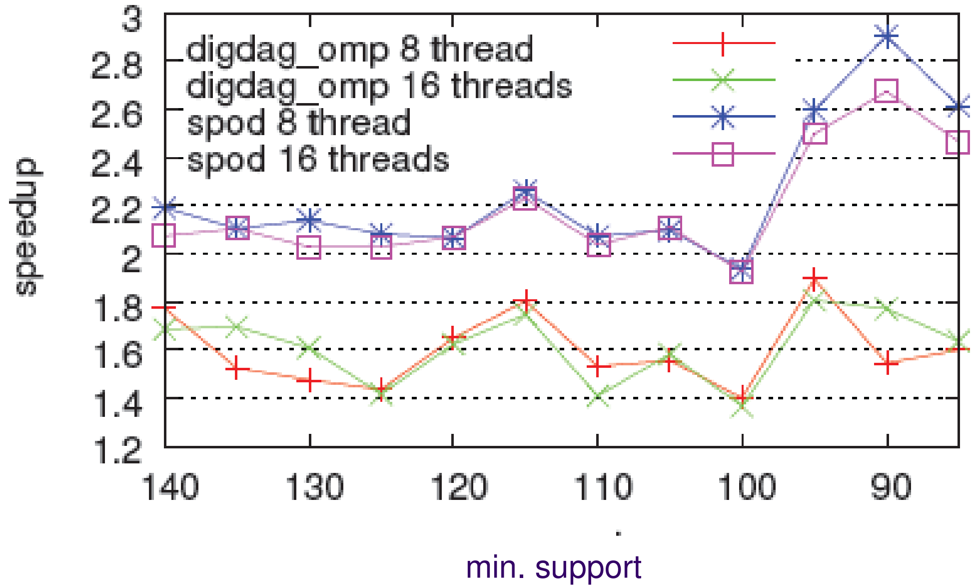




Real data



time (s)
1400
1200
1000
800
600
400
200
0
140
130
120
110
100
90
min. support
digdag sequentiel
digdag_omp 8 thread
digdag_omp 16 threads
spod 8 thread
spod 16 threads



speedup
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
1.2
140
130
120
110
100
90
min. support
digdag_omp 8 thread
digdag_omp 16 threads
spod 8 thread
spod 16 threads

ChronoViz

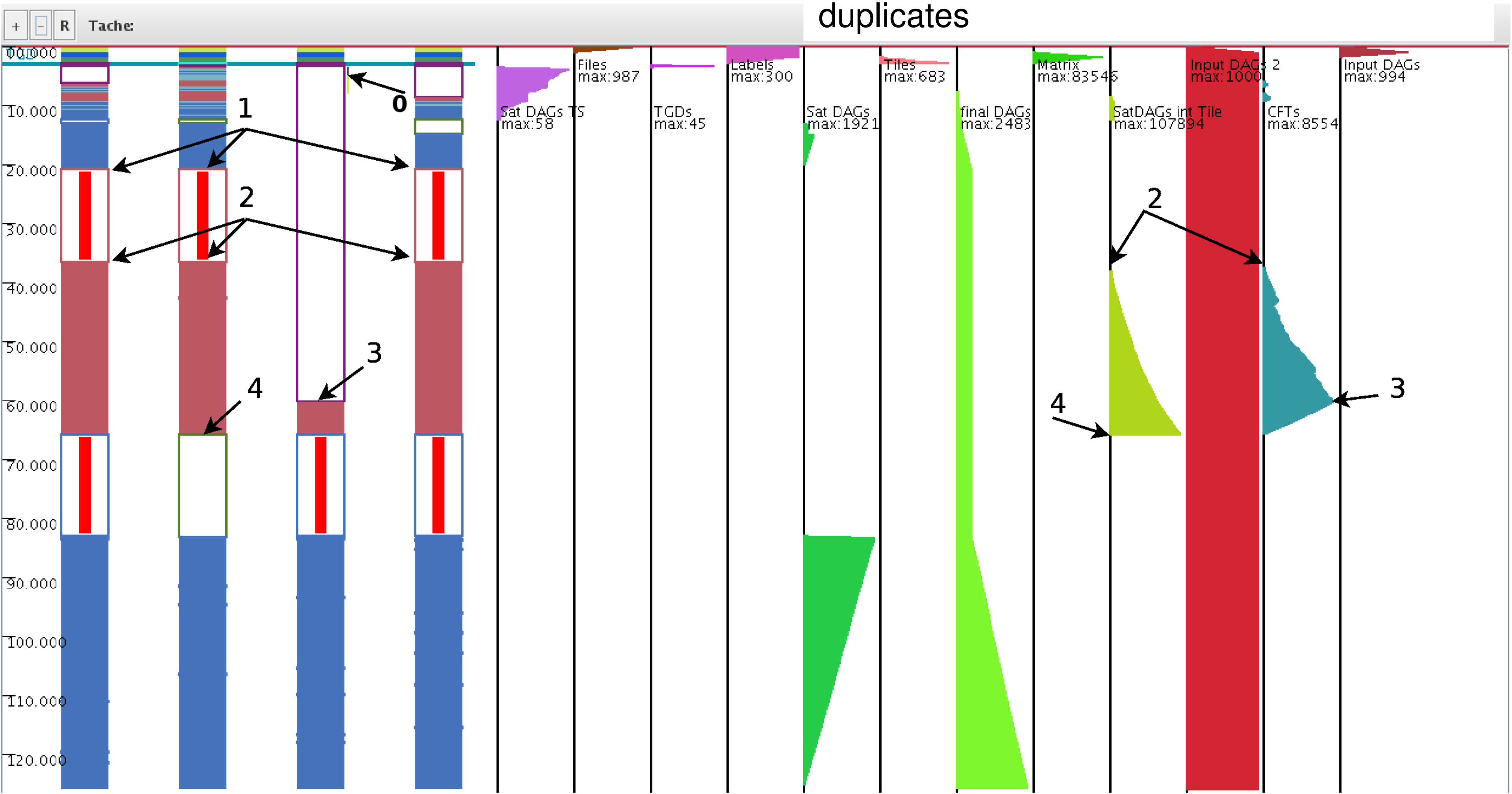
0 : Start consumption of TGD to produce CFT

1 : All threads except T3 stalled

2 : T3 dumps CFTs in TS

3 : T3 has finished working on its TGD

4 : SatDAGs finished : T2 removes duplicates



Conclusion

- Frequent pattern mining
  - Lots of commercial and scientific uses
  - Extremly computation-hungry
- Need of parallel algorithms
- Need of environments for data-miners to write easily parallel algorithms
  - While focusing on the algorithm, not the parallel machinery !
  - → DSL (Domain Specific Language) for frequent pattern mining

Perspectives

- Thesis of B. Negrevergne on the Melinda environment
- Functional languages
  - Problem: performances must be on par with C/Java code
  - Memory handling
- Large scale multicore processors (e.g. Terascale) will allow broad use of Data Mining (cache optimisation, user personalization,…)
  - → lots of fascinating research to come !

ANNEX

Special case: Vertical Data-mining (Zaki *et al.*, Eclat, 1997)

- When #items >> #transactions

	A	B	C	D	E	F	G	H
1								
2								

- Explore transposed matrix

	1	2
A		
B		
C		
D		
E		
F		
G		
H		

ITRS Roadmap

http://www.itrs.net

(International Technology Roadmap for Semiconductors)

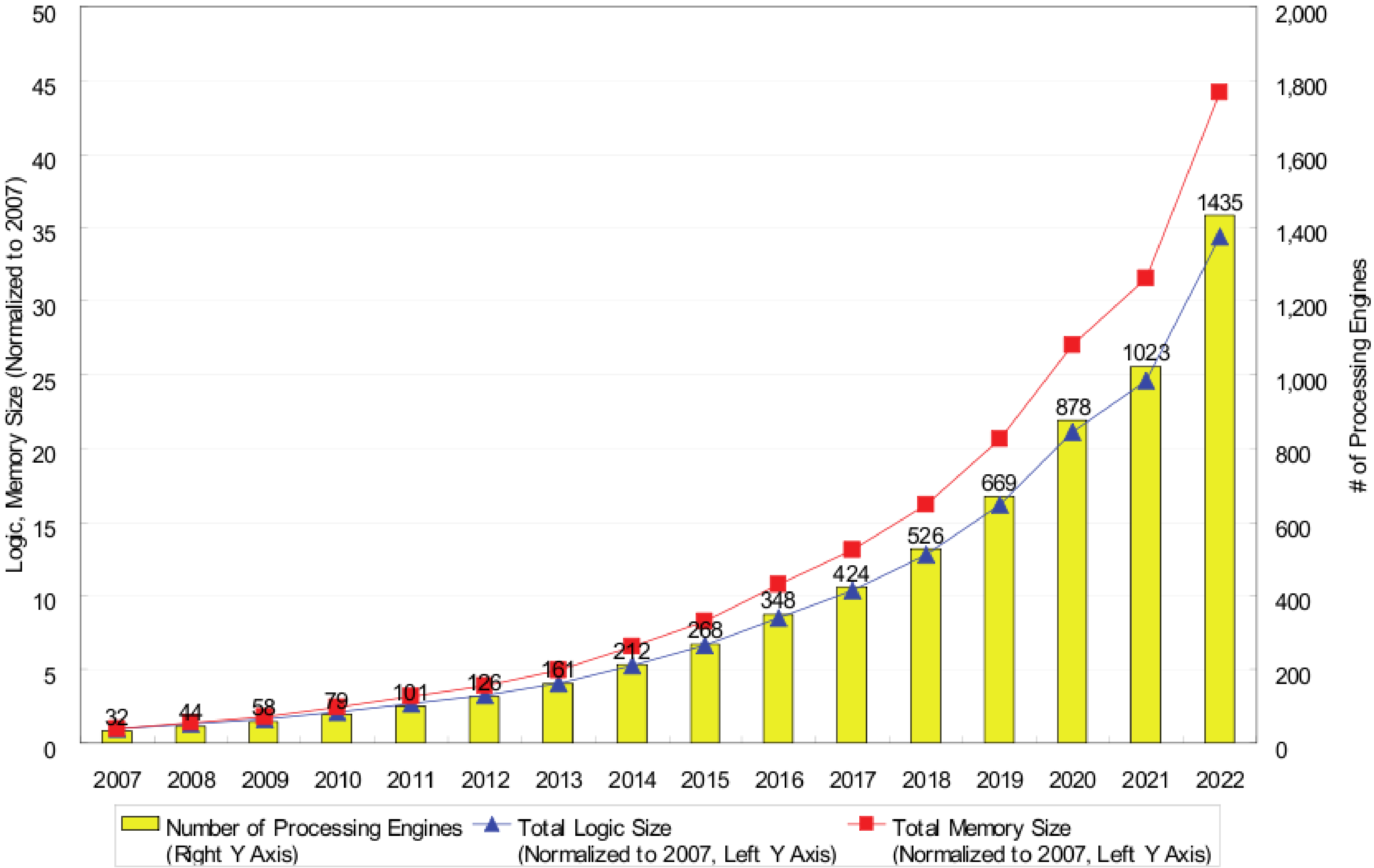


Figure SYSD5 SOC Consumer Portable Design Complexity Trends