A Parallel Pattern Mining Algorithm for Multi-Core Architectures

Benjamin Negrevergne

Jury:

- Marie-Christine Rousset
  LIG/Grenoble University
- Alexandre Termier
  LIG/Grenoble University
- Jean-François Méhaut
  LIG/Grenoble University
- Hiroki Arimura
  Graduate School of Science and Technology/Hokkaido University
- Bruno Crémilleux
  GREYC/University of Caen
- Anne Laurent
  LIRMM/University of Montpellier

Pattern mining

Pattern mining is a sub-topic of **Data Mining**
Knowledge discovery in databases [Fayyad, 96]

"Identifying valid, novel, potentially useful, and ultimately understandable patterns in data"

Pattern mining:

- Extract knowledge as patterns representing regularities (or irregularities) in data.

Broad range of applications

- **Mining frequent set of items purchased together**
  - ▷ _____
- **Mining frequent sub molecules in a molecular database**
  - ▷ _____
- **Mining correlations in sensors data**
  - ▷ _____

Broad range of applications

■ Mining frequent set of items purchased together

▽

Possible frequent pattern: $\{milk, cereals\}$

- Dataset: set of receipts
- Patterns: set of items

■ Mining frequent sub molecules in a molecular database

▷

■ Mining correlations in sensors data

▷

Broad range of applications

- **Mining frequent set of items purchased together**
  - ▷ _____
- **Mining frequent sub molecules in a molecular database**
  - ▷ _____
- **Mining correlations in sensors data**
  - ▷ _____

Broad range of applications

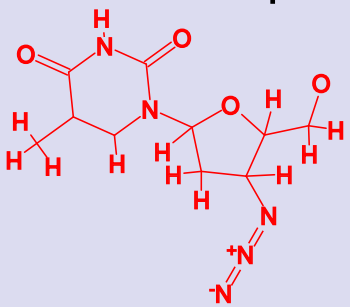
■ Mining frequent set of items purchased together

▷

■ Mining frequent sub molecules in a molecular database

▽

Krammer [2001] was able to identify the following molecular pattern in a set of anti-HIV molecules



- Dataset: set of molecules (represented as graphs)
- Patterns: graphs

Broad range of applications

- **Mining frequent set of items purchased together**
  - ▷ _____
- **Mining frequent sub molecules in a molecular database**
  - ▷ _____
- **Mining correlations in sensors data**
  - ▷ _____

Broad range of applications

- **Mining frequent set of items purchased together**

▷

- **Mining frequent sub molecules in a molecular database**

▷

- **Mining correlations in sensors data**

▽

Records from meteorological sensors
Frequent pattern:
Temperature $\downarrow$ Pressure $\downarrow$ Windspeed $\uparrow$

- Dataset: list of numerical sensor records
- Patterns: set of variations

Broad range of applications

- **Mining frequent set of items purchased together**
  - ▷ _____
- **Mining frequent sub molecules in a molecular database**
  - ▷ _____
- **Mining correlations in sensors data**
  - ▷ _____

The broad range of solutions

Nowadays: many different algorithms for each pattern mining problem

- **Frequent itemset mining**
  - Apriori [Agrawal 94]
  - Eclat [Zaki 1997]
  - FP-Growth [Han 2004]
  - DCI-Closed [Lucchese 2004]
  - LCM [Uno 2004]
- **Graph Mining**
  - AGM [Inokuchi 2000]
  - gSpan [Yan 2002]
  - FSG [Kuramochi 2001]
  - CloseGraph [Yan 2003]
  - Gaston [Nijssen 2004]
- **Tree Mining**
  - FreqT [Asai 2002]
  - TreeMiner [Zaki 2005]
  - Dryade [Termier 2004]
  - CMTreeMiner [Chi 2004]
- **Gradual itemset mining**
  - Grite [Di Jorio 2009]
  - PGP-mc [Laurent 2010]
  - PGLCM [Do 2010]
- **Sequence**
  - Spade [Zaki 2000]
  - PrefixSpan [Pei 2000]
  - CloSpan [Yan 2003]
  - LCM_seq [Uno]

Challenge #1: Genericity

This algorithmic diversity

1. data owners refrain using pattern mining techniques
2. use inadequate algorithms

Challenge #1:

Use a **generic** approach to address different pattern mining problems with a unique algorithm.

Challenge #2: Build an efficient and generic pattern mining algorithm

Pattern mining is inherently **combinatorial**:

- very large number of possible patterns

Example

basket market analysis: 1000 items $\rightarrow 2^{1000}$ ($\sim 10^{300}$) possible patterns.

To be efficient algorithms exploit problem properties to reduce the search space ▶ efficient but non-generic

Challenge #2

Design an **efficient** generic and pattern mining algorithm

Contributions of this thesis

We proposed ParaMiner: both **generic** and **efficient** algorithm
How:

- Extend theoretical work on pattern enumeration [Boley 2007] and [Arimura 2009]
- Tackle large real world datasets through *dataset reduction*
- Exploit parallelism for multi-core architectures

Contributions of this thesis

We proposed ParaMiner: both **generic** and **efficient** algorithm
How:

- Extend theoretical work on pattern enumeration [Boley 2007] and [Arimura 2009]
- Tackle large real world datasets through *dataset reduction*
- Exploit parallelism for multi-core architectures

Results

ParaMiner:

- solves various pattern mining problems
- is time-efficient (compete with ad-hoc algorithms)

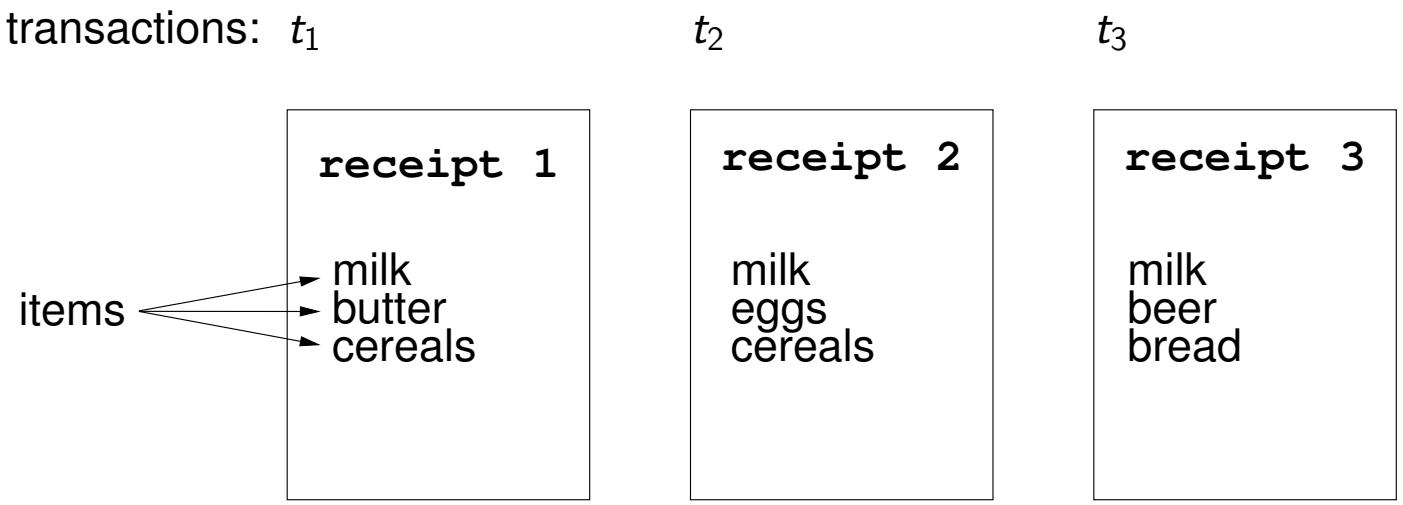
Outline

Illustration: frequent itemset mining

receipt 1	receipt 2	receipt 3
milk	milk	milk
butter	eggs	beer
cereals	cereals	bread

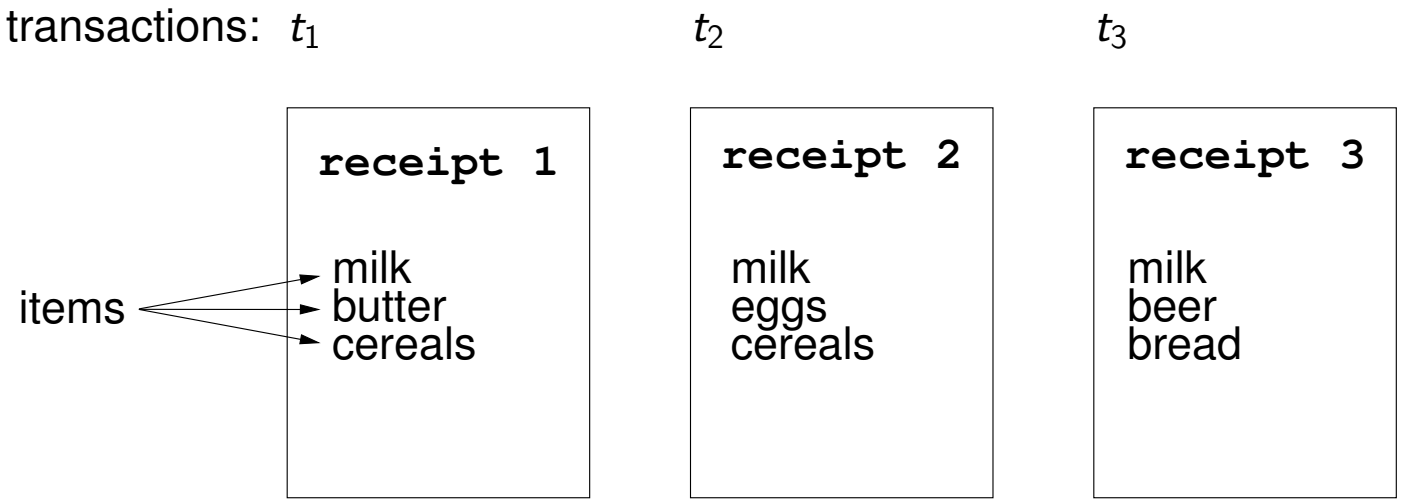
minimum frequency threshold $= 50\%$

Illustration: frequent itemset mining



minimum frequency threshold $= 50\%$

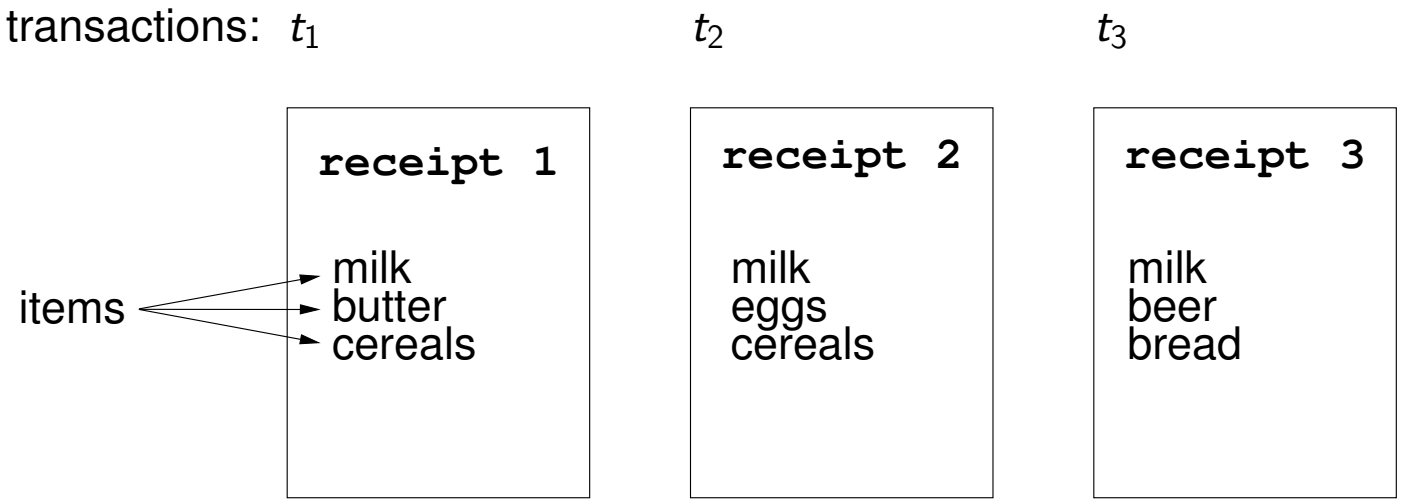
Illustration: frequent itemset mining



minimum frequency threshold = 50%

- $P_1 = \{milk\}$ is frequent (100%): t_1, t_2, t_3

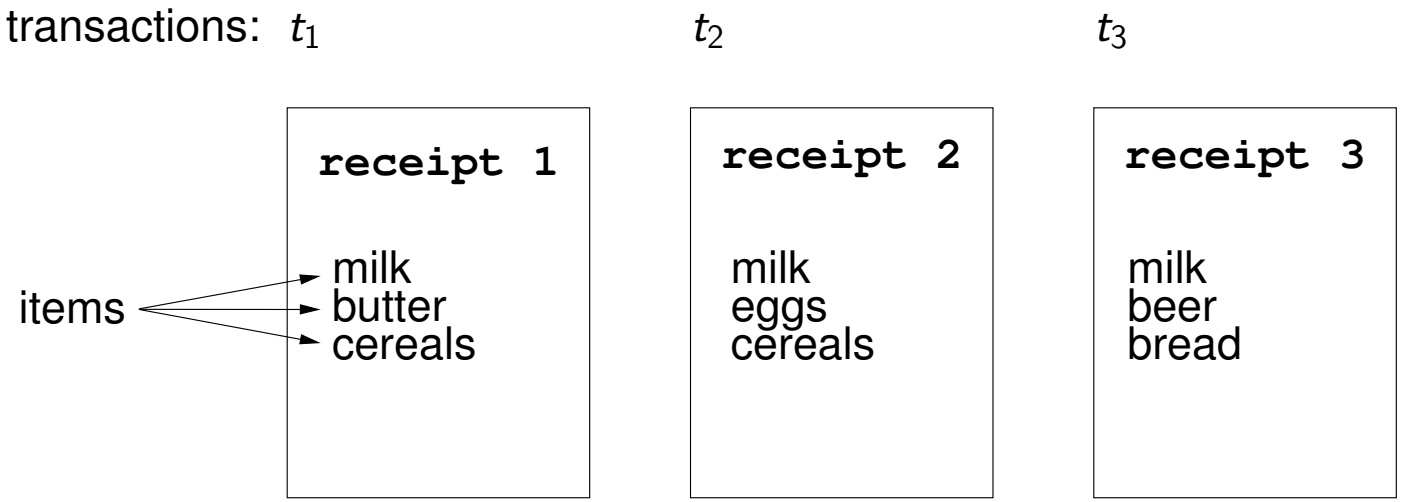
Illustration: frequent itemset mining



minimum frequency threshold = 50%

- $P_1 = \{milk\}$ is frequent (100%): t_1, t_2, t_3
- $P_2 = \{cereals\}$ is frequent (66%): t_1, t_2

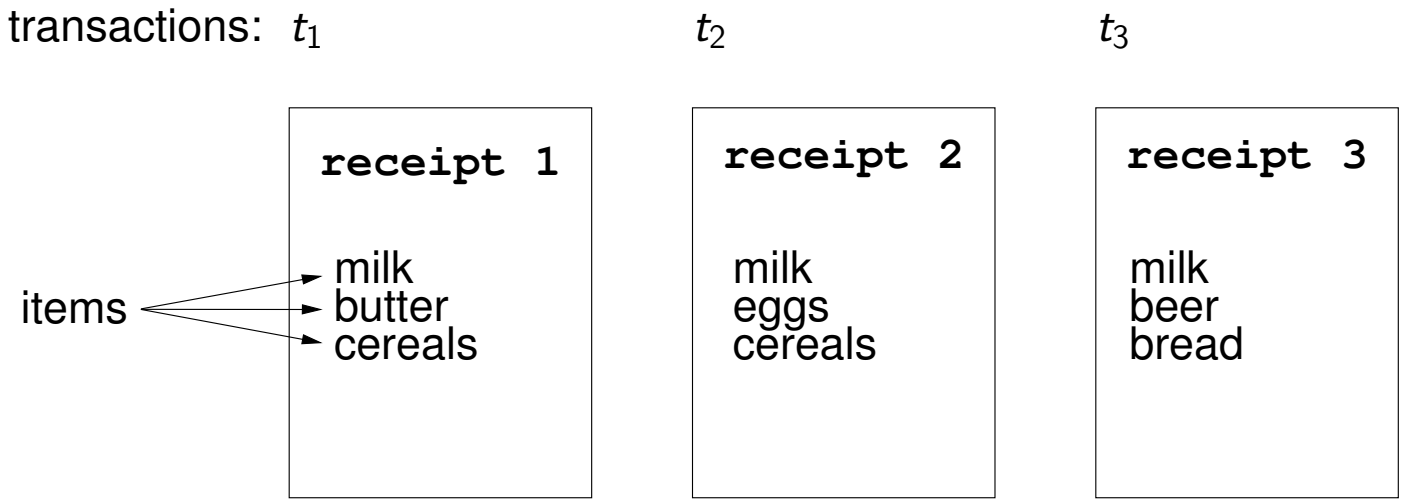
Illustration: frequent itemset mining



minimum frequency threshold = 50%

- $P_1 = \{milk\}$ is frequent (100%): t_1, t_2, t_3
- $P_2 = \{cereals\}$ is frequent (66%): t_1, t_2
- $P_3 = \{milk, cereals\}$ is frequent (66%): t_1, t_2

Illustration: frequent itemset mining



minimum frequency threshold = 50%

- $P_1 = \{milk\}$ is frequent (100%): $t_1, t_2, t_3 \rightarrow$ closed
- $P_2 = \{cereals\}$ is frequent (66%): t_1, t_2
- $P_3 = \{milk, cereals\}$ is frequent (66%): $t_1, t_2 \rightarrow$ closed

Closed pattern: maximal pattern in a set of transactions

Generic framework

- Every pattern represented as a set

- A pattern mining problem defined
  - A ground set E
  - A dataset $\mathcal{D}_E$
  - A selection criterion *Select*

Ground set

Definition

- Set of all possible elements
- Every candidate pattern is a **subset** of the ground set

Ground set

Definition

- Set of all possible elements
- Every candidate pattern is a **subset** of the ground set

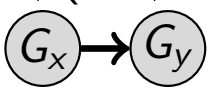
Examples

Frequent itemsets ▶ E: **all possible items**

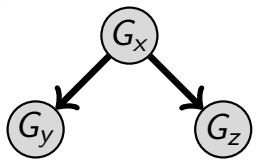
$E = \{milk, cereals, eggs, \ldots\}$ e.g. candidate: $\{milk, eggs\}$

Connected relational graphs ▶ E: **all the possible arcs**

$E = \{(G_1, G_2), (G_1, G_3), \ldots, (G_5, G_4)\}$

(G_x, G_y) represent the arc 

eg. candidate:



Gradual itemsets ▶ E: **all the possible variations**

$E = \{T\uparrow, T\downarrow, P\uparrow, \ldots, W\downarrow\}$ eg. candidate: $\{T\uparrow, W\uparrow\}$

Dataset

Definition

- Sequence of *transactions* $\mathcal{D}_E$
- Each **transaction** is a subset of E

Dataset

Definition

- Sequence of *transactions* $\mathcal{D}_E$
- Each **transaction** is a subset of E

Examples

Frequent itemsets ▶ in $\mathcal{D}_E$ **each transaction = receipt**

Dataset

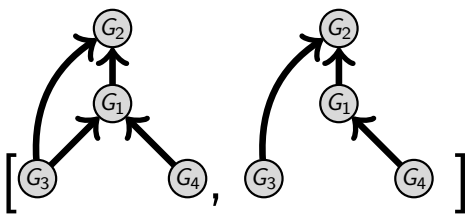
Definition

- Sequence of *transactions* $\mathcal{D}_E$
- Each **transaction** is a subset of E

Examples

Frequent itemsets ▶ in $\mathcal{D}_E$ **each transaction = receipt**

Relational graphs ▶ in $\mathcal{D}_E$ **each transaction = input graph**



$$\Rightarrow \begin{array}{l}[\{(G_1, G_2), (G_3, G_1), (G_3, G_2), (G_4, G_1)\}, \\ \{(G_1, G_2), (G_3, G_2), (G_4, G_1)\}]\end{array}$$

Dataset (2)

Gradual itemsets ▶ in $\mathcal{D}_E$ **each transaction = variation of attributes between pairs of records**

Date	T	P	W
Day 1	17.6	1021.20	57
Day 2	18.5	1021.30	56
Day 3	20.4	1018.20	51

$\Rightarrow$

(d_1, d_2)	$[\{T\uparrow, P\uparrow, W\downarrow\}$
(d_1, d_3)	$\{T\uparrow, P\downarrow, W\downarrow\}$
(d_2, d_1)	$\{T\downarrow, P\downarrow, W\uparrow\}$
(d_2, d_3)	$\{T\uparrow, P\downarrow, W\downarrow\}$
(d_3, d_1)	$\{T\downarrow, P\uparrow, W\uparrow\}$
(d_3, d_2)	$\{T\downarrow, P\uparrow, W\uparrow\}]$

Selection criterion

Definition

- User-defined predicate $2^E \rightarrow \{true, false\}$
- $Select(P, \mathcal{D}_E) \equiv P$ *is a* **pattern** *of interest in* $\mathcal{D}_E$

Selection criterion

Definition

- User-defined predicate $2^E \rightarrow \{true, false\}$
- $Select(P, \mathcal{D}_E) \equiv P$ *is a* **pattern** *of interest in* $\mathcal{D}_E$

Examples

Frequent itemsets ▶ $Select(P, \mathcal{D}_E) \equiv P$ is frequent in $\mathcal{D}_E$

Connected relational graphs ▶ $Select(P, \mathcal{D}_E) \equiv P$ is a connected graph **and** P is frequent in $\mathcal{D}_E$

Gradual itemsets ▶ $Select(P, \mathcal{D}_E) \equiv$ there exists a path $[(d_{x_1}, d_{x_2}), \ldots, (d_{x_{n-1}}, d_{x_n})]$ with $n \geq minsup$

General pattern mining problem

Closed patterns

Closed pattern: Any subset P of E such that

- P occurs in $\mathcal{D}_E$ ($\mathcal{D}_E[P] \neq \emptyset$)
- P satisfies the selection criterion
- P is the maximal pattern in $\mathcal{D}_E[P]$

Problem statement

Given a **ground set** E, **a dataset** $\mathcal{D}_E$ and a **selection criterion**

▶ extract all the **closed patterns**

Outline

The generate and test principle

The standard approach

generate and test.

1. **Generate** candidate patterns
2. **Test** if the candidate pattern is a pattern

The generate and test principle

The standard approach

generate and test.

1. **Generate** candidate patterns
2. **Test** if the candidate pattern is a pattern

. . . is not naively applicable

- **Too many** candidate patterns
- **Each test** requires costly database accesses
  (e.g. to count frequency)

▶ Challenges

- Generic strategy to explore the set of candidate patterns
- Generic method to simplify the testing

Outline

Efficient exploration of the set of candidate patterns

Structured search space

Most algorithms define a **pattern augmentation relation**

- search space is explored by repeatedly augmenting patterns

ParaMiner's augmentation relation

P and Q, two patterns:

- Q is the augmentation of $P \Leftrightarrow Q = P \cup \{e\}$

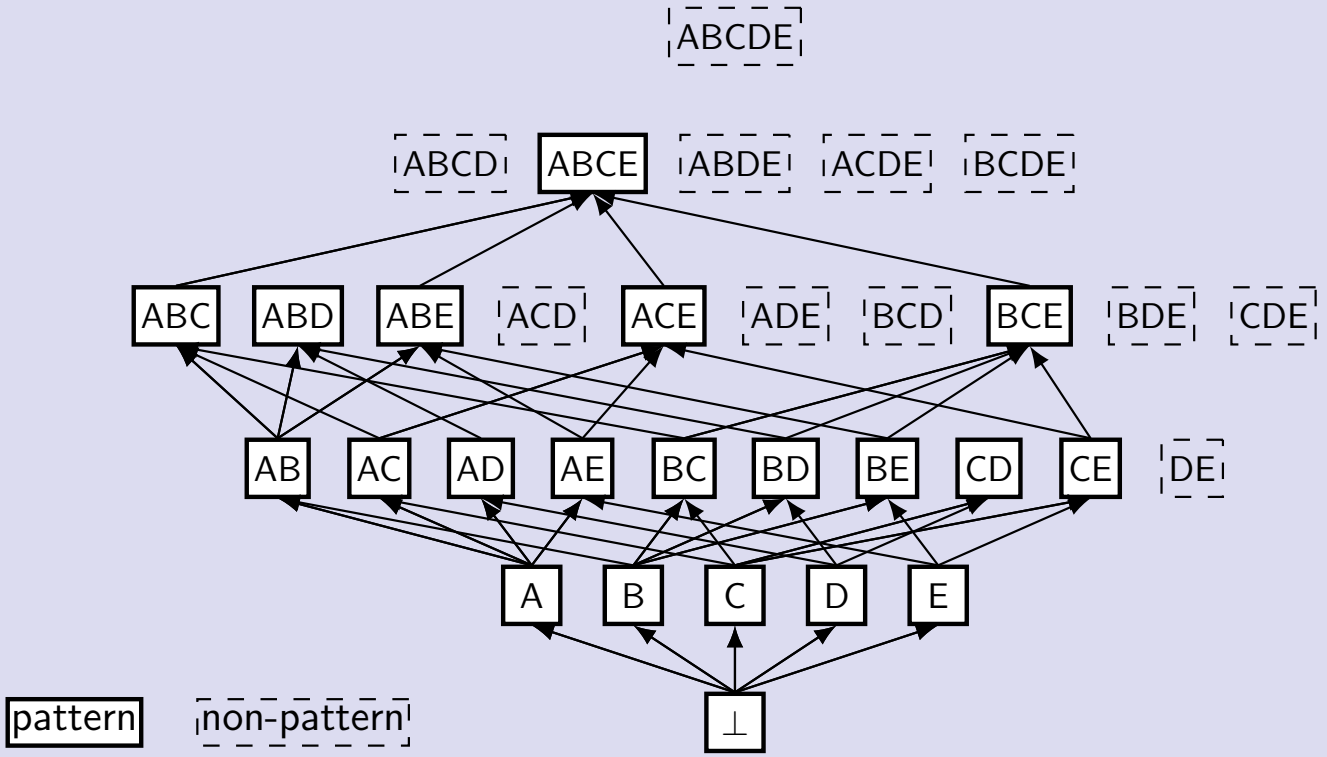
$$\boxed{\text{ACE}} \rightarrow \boxed{\text{ABCE}}$$

Set of patterns + Augmentation relation = DAG[1] structure

[1] *directed acyclic graph*

Structured search space

DAG-structure of a set of patterns



- several arcs lead to the same pattern

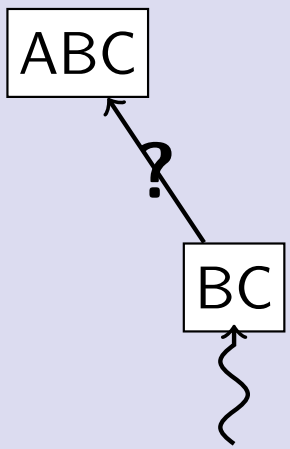
Discover the structured search space

An enumeration strategy

Is required to:

- discover all the patterns
- avoid **duplicates generation**

Which augmentation path must be followed ?



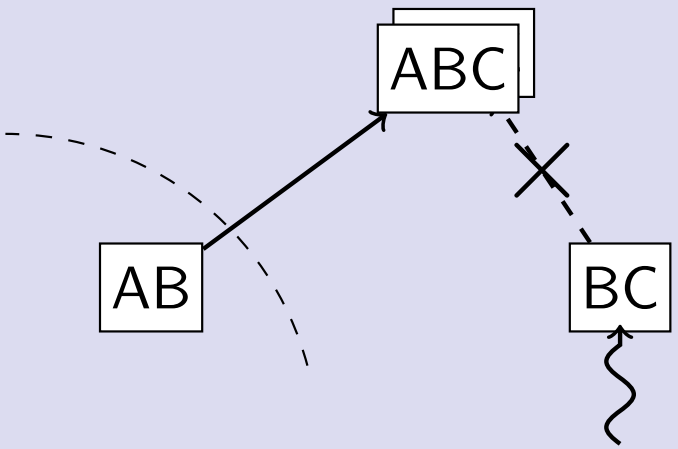
Discover the structured search space

An enumeration strategy

Is required to:

- discover all the patterns
- avoid **duplicates generation**

Which augmentation path must be followed ?



▶ AB is the **first_parent** of ABC

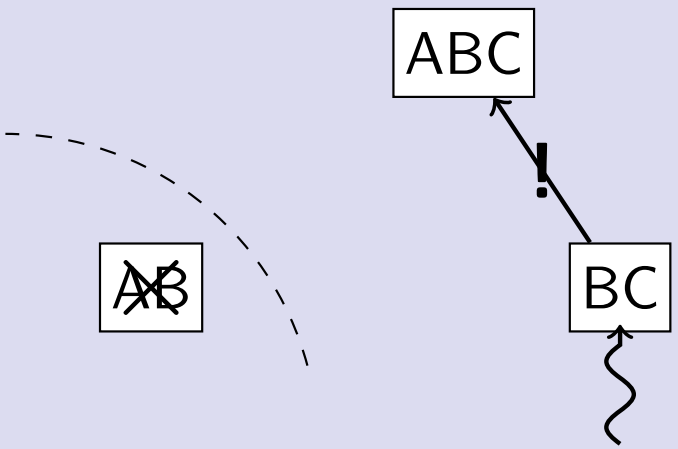
Discover the structured search space

An enumeration strategy

Is required to:

- discover all the patterns
- avoid **duplicates generation**

Which augmentation path must be followed ?



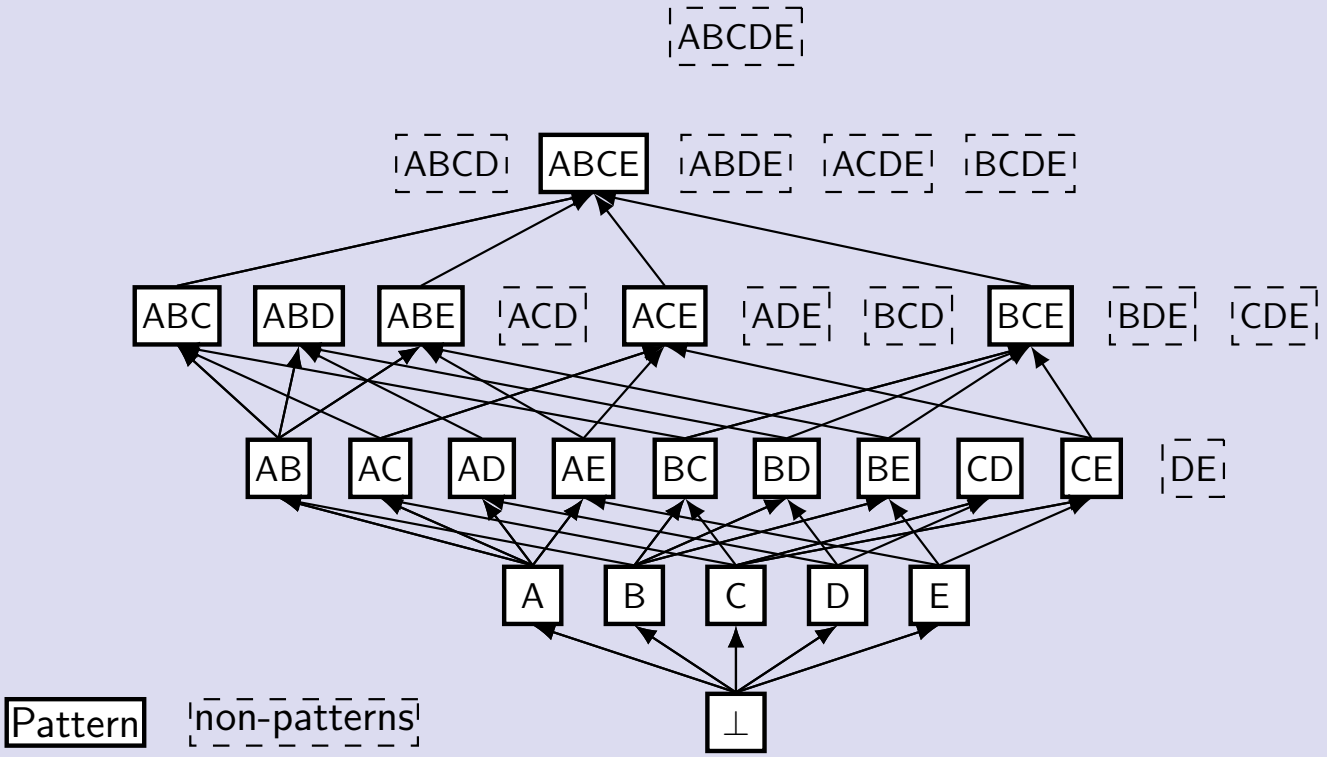
▶ BC is the **first_parent** of ABC

Enumeration strategy

Follow the augmentation $\boxed{P} \rightarrow \boxed{Q}$ if:

first_parent$(\boxed{Q}) = \boxed{P}$

DAG-structure following a tree search

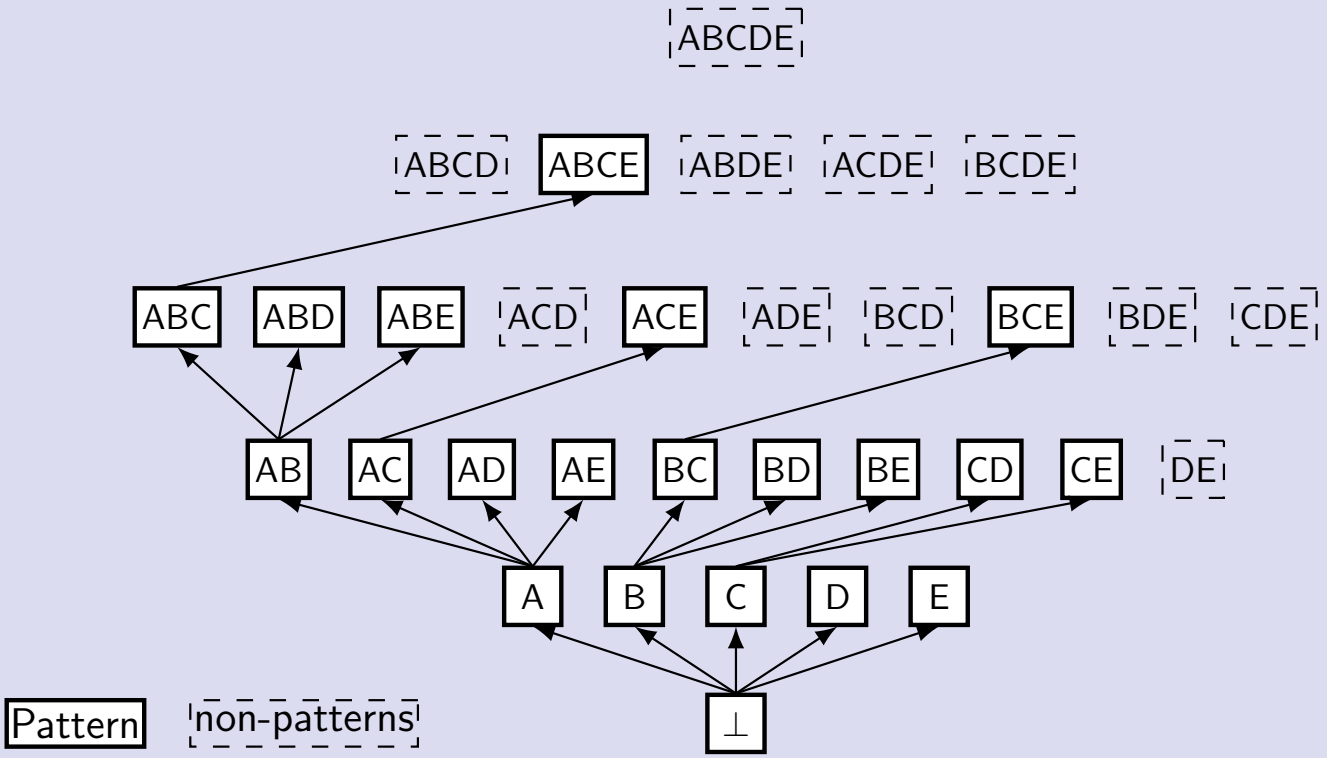


Enumeration strategy

Follow the augmentation $\boxed{P} \rightarrow \boxed{Q}$ if:

first_parent$(\boxed{Q}) = \boxed{P}$

DAG-structure following a tree search



How to compute the first parent of a pattern

Requirement

A method that is:

- **Generic**
  - ▶ Sound for all our pattern mining problems
- **Adequate to parallel exploration of the search space**
  - ▶ Computable without **global computations**

How to compute the first parent of a pattern

Requirement

A method that is:

- **Generic**
  - ▶ Sound for all our pattern mining problems
- **Adequate to parallel exploration of the search space**
  - ▶ Computable without **global computations**

State of the art:

Theoretical work closed pattern enumeration strategies

- [Arimura et al. 2009]
- [Boley et al. 2010]

▶ Poly-space enumeration strategies

▶ first_parent do not rely on a global representation of the search space

Structural properties of the set of patterns

[Boley 2010] **Strong accessibility**:

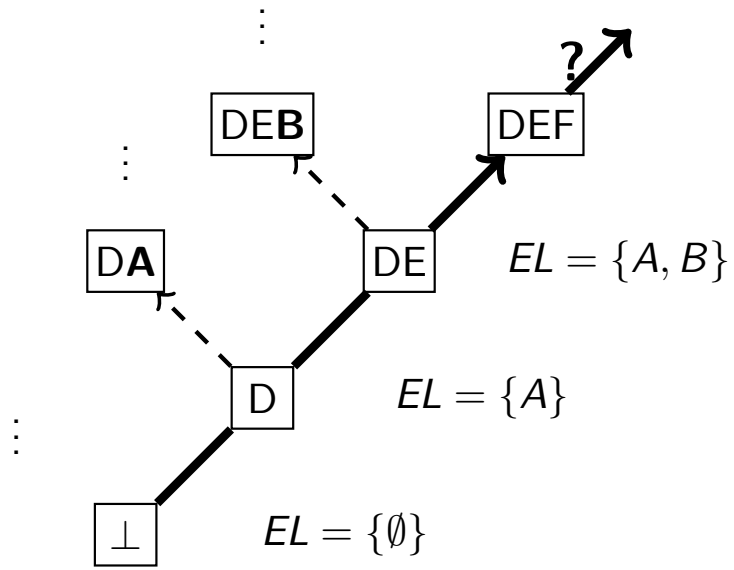
Idea: *We can reach every pattern by augmenting any subset that is a pattern*

Structural properties of the set of patterns

[Boley 2010] **Strong accessibility**:

Idea: *We can reach every pattern by augmenting any subset that is a pattern*

We can detect first parent by memorizing **only the root** of the forking branches:



- Branch can be explored independently → Parallel
- FIM, CRG and GRI satisfy this property (proved) → Generic

Conclusion on pattern enumeration

- Pattern augmentation
- Structured search space
- Enumeration strategy required
- Problem of detecting the first parent
- Strong accessibility property
- Exclusion list

▶ generic and parallelizable enumeration strategy

Next: Identifying patterns

Outline

Simplifying candidate pattern testing

Candidate pattern testing

- Check candidate pattern occurrence in the dataset
- Computing the selection criterion
- Computing pattern closure

▶ Intensive access to the dataset

Problem: How to reduce dataset accesses ?

▶ dataset reduction [Han 2000, Uno 2004]

Dataset reduction: principle

Motivation: Every element in the dataset are not required to test each candidate pattern

Principle:

1. build a dataset **for each** new pattern P
2. **filter** unnecessary elements to P and its descendants
3. use it to test P and its descendants

In ParaMiner : EL-based filter ▶ generic

EL-based reduction (Proved)

Dataset:

$\mathcal{D}_E$

Pattern:

$P = \{\text{D, E}\}$

EL:

$EL = \{\text{A, B, C}\}$

$\mathcal{D}_E$	
t_1	A, B, D, E, F, G,
t_2	A, D, D, E, F, G,
t_3	A, B, D, E, F, H,

- $\mathcal{D}_P^{reduced}$?

EL-based reduction (Proved)

Dataset:
$\mathcal{D}_E$
Pattern:
$P = \{\text{D, E}\}$
EL:
$EL = \{\text{A, B, C}\}$

$\mathcal{D}_E$	$\in EL$	P	$augm.$ $(\notin EL)$
t_1	A, B,	D, E,	F, G,
t_2	A, D,	D, E,	F, G,
t_3	A, B,	D, E,	F, H,

- $\mathcal{D}_P^{reduced}$?
  - Element required: Any e that can belongs to a closed pattern including P ▶ ensure that first_parent is sound

EL-based reduction (Proved)

Dataset:
$\mathcal{D}_E$
Pattern:
$P = \{\text{D, E}\}$
EL:
$EL = \{\text{A, B, C}\}$

$\mathcal{D}_E$	$\in EL$	P	$augm.$ $(\notin EL)$
t_1	A, B,	D, E,	F, G,
t_2	A, D,	D, E,	F, G,
t_3	A, B,	D, E,	F, H,

- $\mathcal{D}_P^{reduced}$?
  - Element required: Any e that can belongs to a closed pattern including P ▶ ensure that first_parent is sound

EL-based reduction (Proved)

Dataset:
$\mathcal{D}_E$
Pattern:
$P = \{D, E\}$
EL:
$EL = \{A, B, C\}$

$\mathcal{D}_E$	$\in EL$	P	augm.	$(\notin EL)$	
t_1	A, B,	D, E,	F, G,		$\mathcal{P}_1$
t_2	A, D,	D, E,	F, G,		$\mathcal{P}_1$
t_3	A, B,	D, E,	F, H,		$\mathcal{P}_2$

- $\mathcal{D}_P^{reduced}$?
  - Element required: Any e that can belongs to a closed pattern including P ▶ ensure that first_parent is sound

How to detect them:

- grouping transactions per augmentations supported
- Apply reduction

EL-based reduction (Proved)

Dataset:
$\mathcal{D}_E$
Pattern:
$P = \{D, E\}$
EL:
$EL = \{A, B, C\}$

$\mathcal{D}_E$	$\in EL$	P	augm.	$(\notin EL)$
t_1	A, B,	D, E,	F, G,	$\mathcal{P}_1$
t_2	A, D,	D, E,	F, G,	$\mathcal{P}_1$
t_3	A, B,	D, E,	F, H,	$\mathcal{P}_2$
		$\Downarrow$		
$\mathcal{D}_P^{reduced}$	$\bigcap$	$=$	$=$	
$t'_1 \oplus t'_2$	A,	D, E,	F, G,	
t'_3	A, B,	D, E,	F, H,	

▶ $\mathcal{D}_P^{reduced}$?

- Element required: Any e that can belongs to a closed pattern including P ▶ ensure that first_parent is sound

How to detect them:

- grouping transactions per augmentations supported
- Apply reduction

Evaluation of dataset reduction

Metric

$$reduction\_factor = \frac{|\mathcal{D}_E|}{|\mathcal{D}_P^{reduced}|}$$

$$average\_reduction\_factor = \frac{\sum_{P \in \mathcal{C}} |\mathcal{D}_E| / |\mathcal{D}_P^{reduced}|}{|\mathcal{C}|}$$

Frequent itemset mining, mushroom dataset

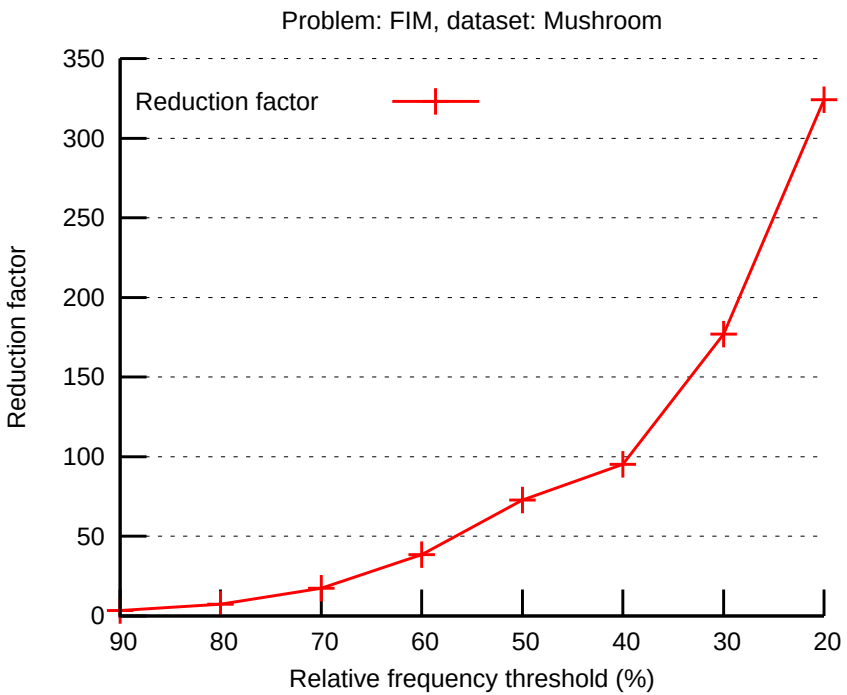
dataset name	ground set size	# transactions	dataset size
Mushroom	119	8,124	186,852

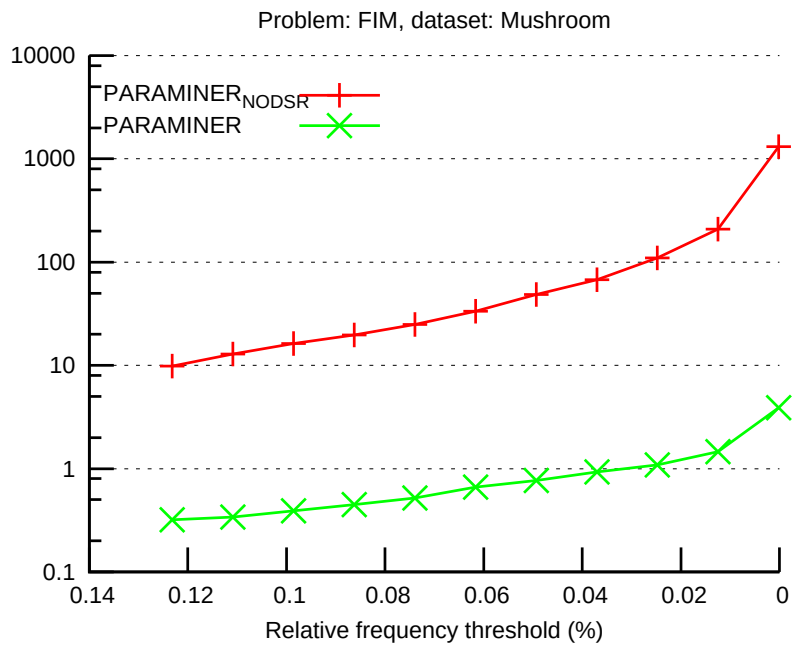
Evaluation of dataset reduction

Metric

$$reduction\_factor = \frac{|\mathcal{D}_E|}{|\mathcal{D}_P^{reduced}|}$$

$$average\_reduction\_factor = \frac{\sum_{P \in \mathcal{C}} |\mathcal{D}_E| / |\mathcal{D}_P^{reduced}|}{|\mathcal{C}|}$$





ParaMiner: algorithm

1: **procedure** **expand**$(P, \mathcal{D}_P^{reduced}, EL)$
2: **for all** e such that e occurs in $\mathcal{D}_P^{reduced}$ **do**
3: **if** $Select(P \cup \{e\}, \mathcal{D}_P^{reduced})$ **then**
4: $Q \leftarrow Clo(P \cup \{e\}, \mathcal{D}_P^{reduced})$
5: **if** $is\_first\_parent(P, EL, Q)$ **then**
6: **output** Q
7: $\mathcal{D}_Q^{reduced} \leftarrow reduce(\mathcal{D}_P^{reduced}, e, EL)$
8: **spawn** expand$(Q, \mathcal{D}_Q^{reduced}, EL)$
9: $EL \leftarrow EL \cup \{e\}$
10: **end if**
11: **end if**
12: **end for**
13: **end procedure**

ParaMiner: main achievements

- **Proved sound and complete** under some constraints:
  - the set of patterns satisfying the **strong accessibility** property
  - the selection criterion is **decomposable**

  Problems FIM, CRG and GRI satisfy these properties

- **Parallelized**: space exploration is divided into independent tasks
  - non-trivial for **closed** pattern mining

ParaMiner: main achievements

- **Proved sound and complete** under some constraints:
  - the set of patterns satisfying the **strong accessibility** property
  - the selection criterion is **decomposable**

  Problems FIM, CRG and GRI satisfy these properties

- **Parallelized**: space exploration is divided into independent tasks
  - non-trivial for **closed** pattern mining

- Melinda a parallel execution engine for **data-driven** algorithms
  - can be extended with task scheduling strategies
  - execute tasks spawned by ParaMiner
  - also used in PLCM [Negrevergne 2010], PGLCM [Do 2010]

Outline

Melinda: Main concepts

- **Tuples** (A, B, C) ▶ block of data with a fixed structure
- **Tuple Space** ▶ synchronized memory space
  - can contain tuples
  - supports **put()** and **get()**

Parallelizing ParaMiner with Melinda

ParaMiner parallelization scheme: A new tuple for each recursive call

Tuple structure: Parameters of recursive calls
$(Pattern, Reduce\_dataset, Exclusion\_list)$

Melinda: Main concepts

- **Tuples** (A, B, C) ▶ block of data with a fixed structure
- **Tuple Space** ▶ synchronized memory space
  - can contain tuples
  - supports **put()** and **get()**

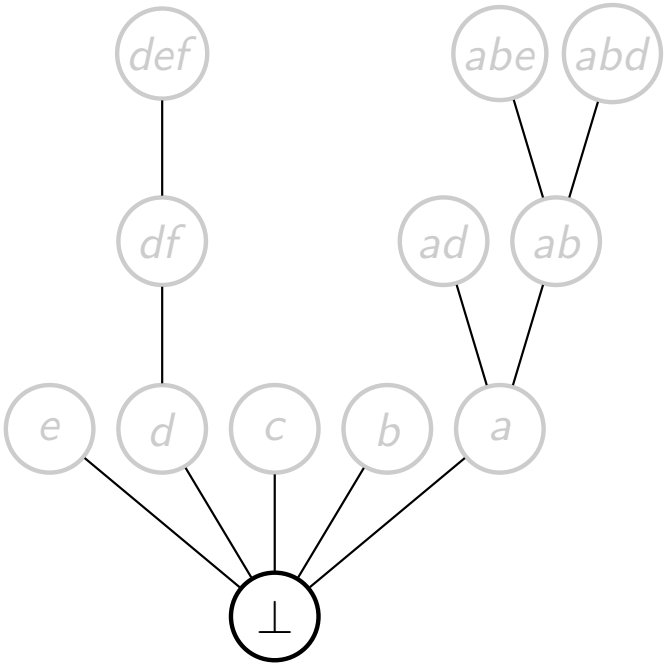
Parallelizing ParaMiner with Melinda

ParaMiner parallelization scheme: A new tuple for each recursive call

Tuple structure: Parameters of recursive calls
($Pattern$, $Reduce\_dataset$, $Exclusion\_list$)

- Tuples are **produced** when a pattern is discovered
- Tuples are **consumed** when a core is idle

Parallel execution



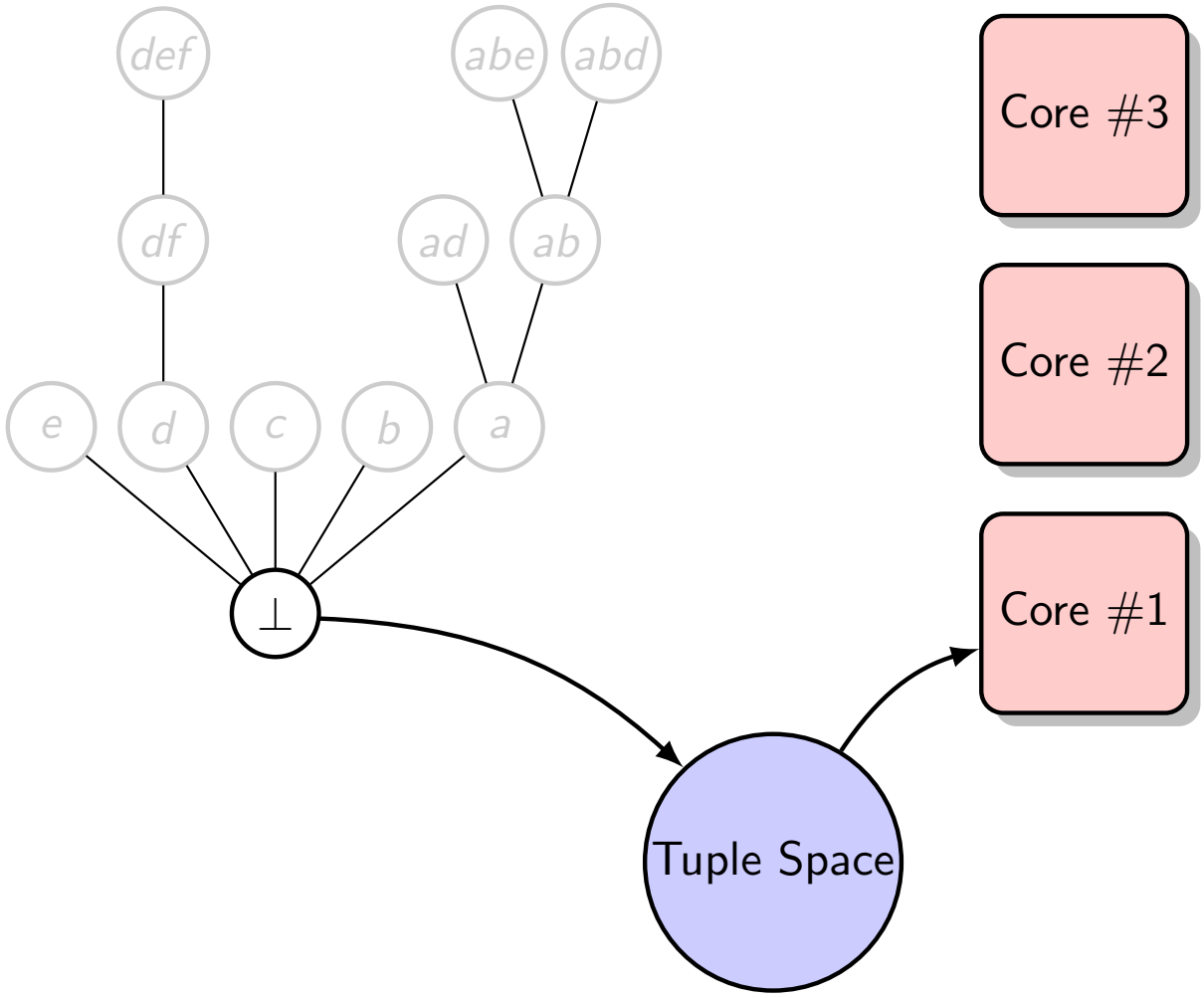
Core #3

Core #2

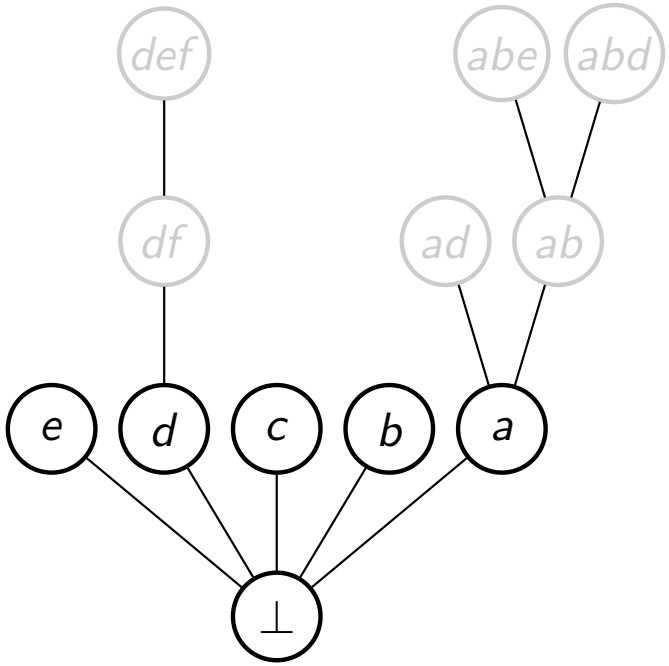
Core #1

Tuple Space

Parallel execution



Parallel execution



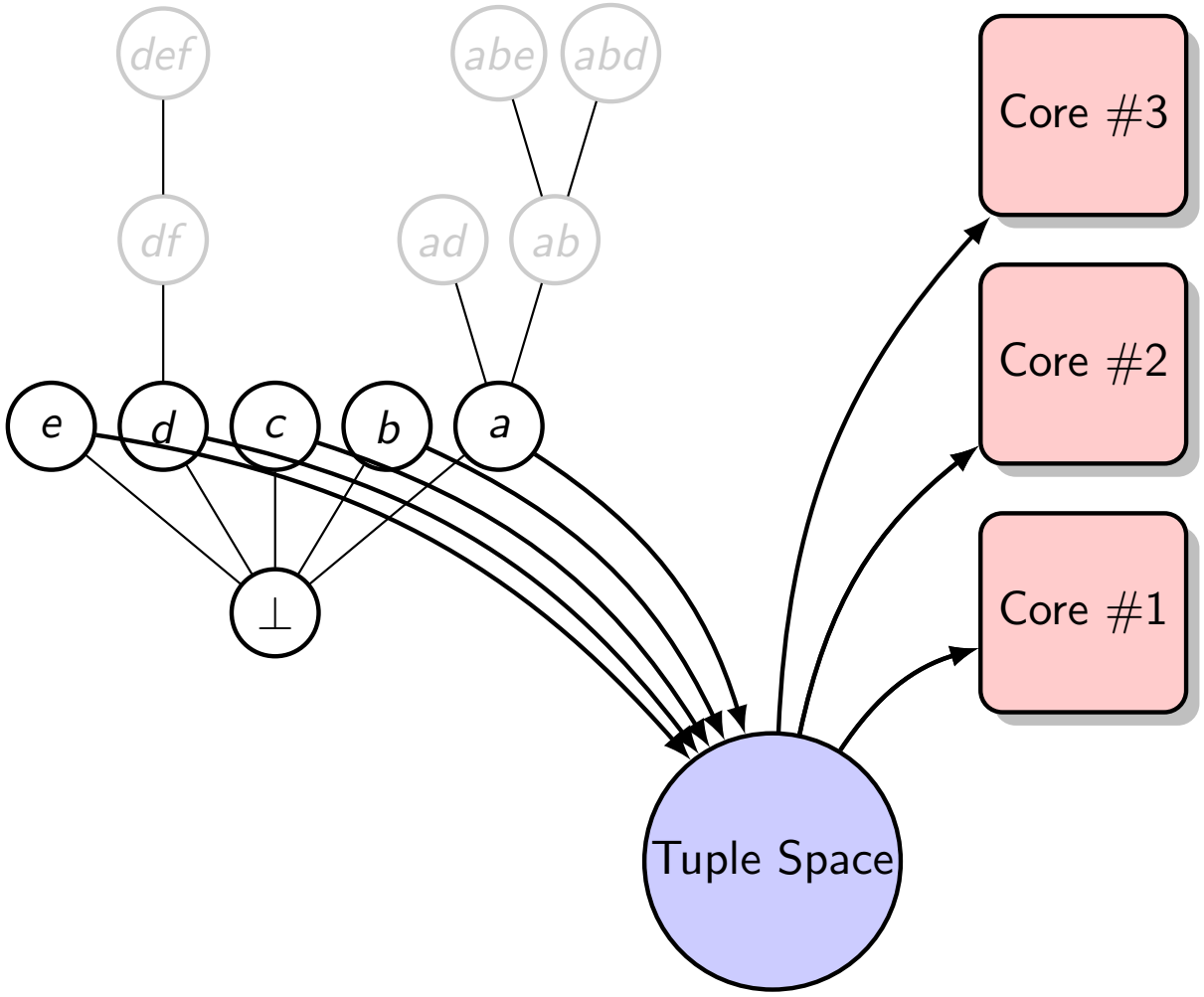
Core #3

Core #2

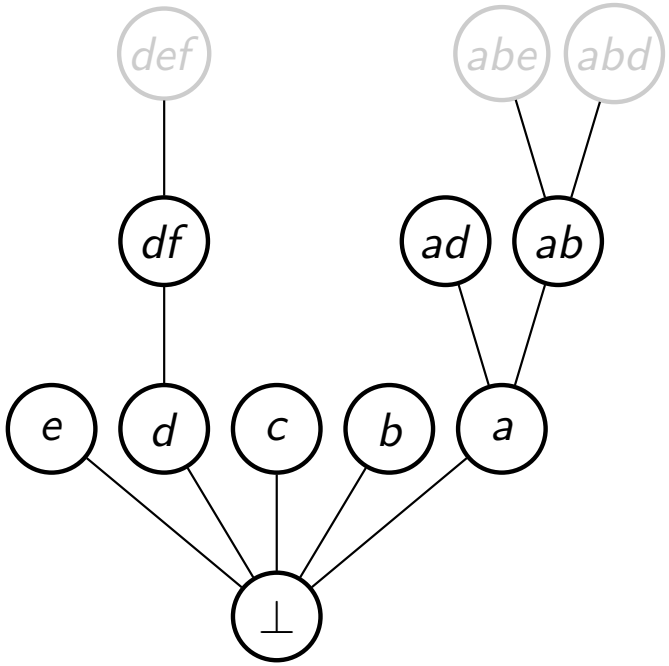
Core #1

Tuple Space

Parallel execution



Parallel execution



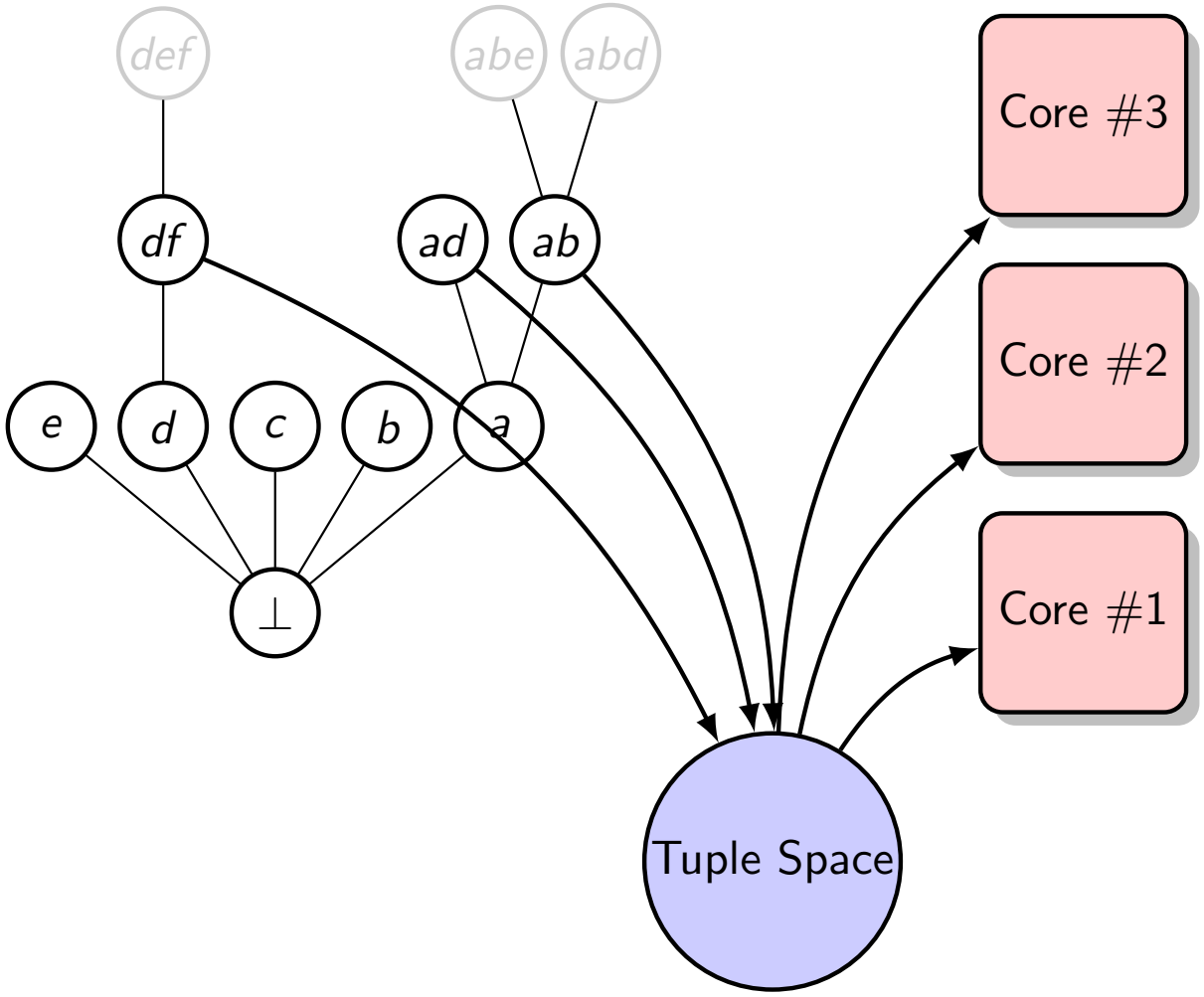
Core #3

Core #2

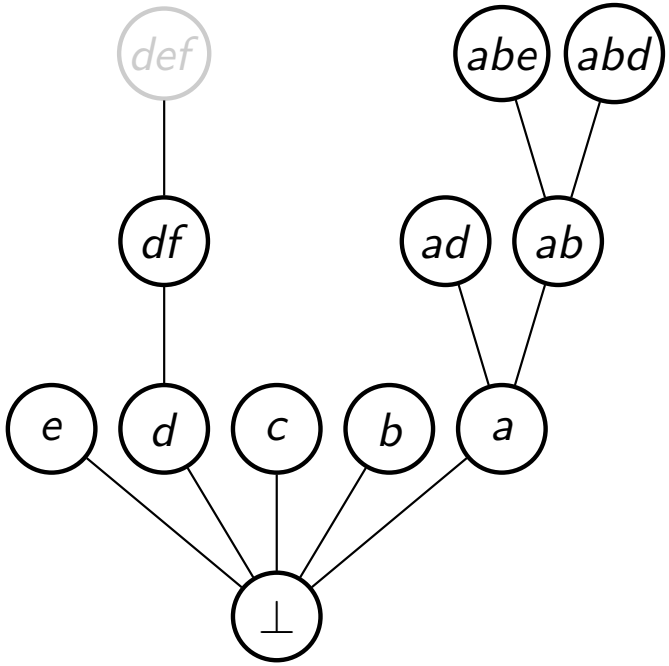
Core #1

Tuple Space

Parallel execution



Parallel execution



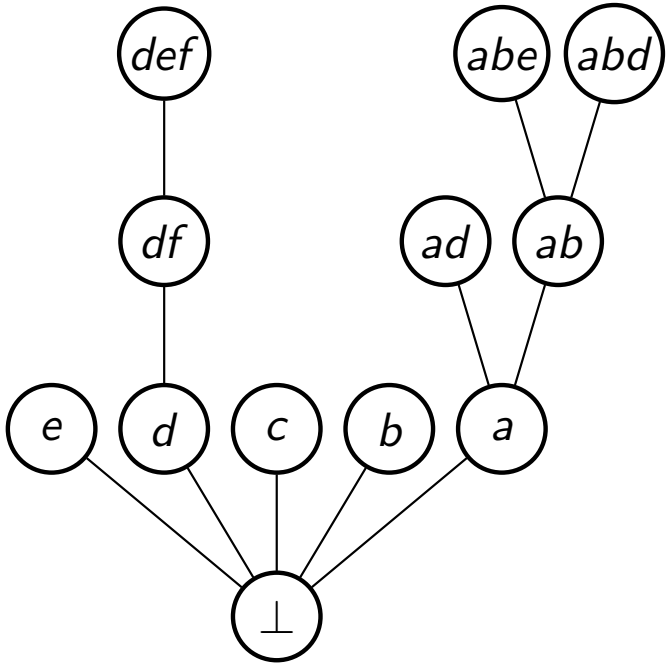
Core #3

Core #2

Core #1

Tuple Space

Parallel execution



Core #3

Core #2

Core #1

Tuple Space

Outline

Outline

Parallel performance evaluation

ParaMiner instances

- frequent itemset mining
- frequent relational graph mining
- gradual itemset mining

Computing platforms

Laptop:

- 4-cores (4 core *i*7)
- 8 GB memory

Server:

- 32-cores (4 core *i*7 with each core each)
- 64 GB memory

Metric

$$speedup = \frac{\text{time using one core}}{\text{time using n cores}}$$

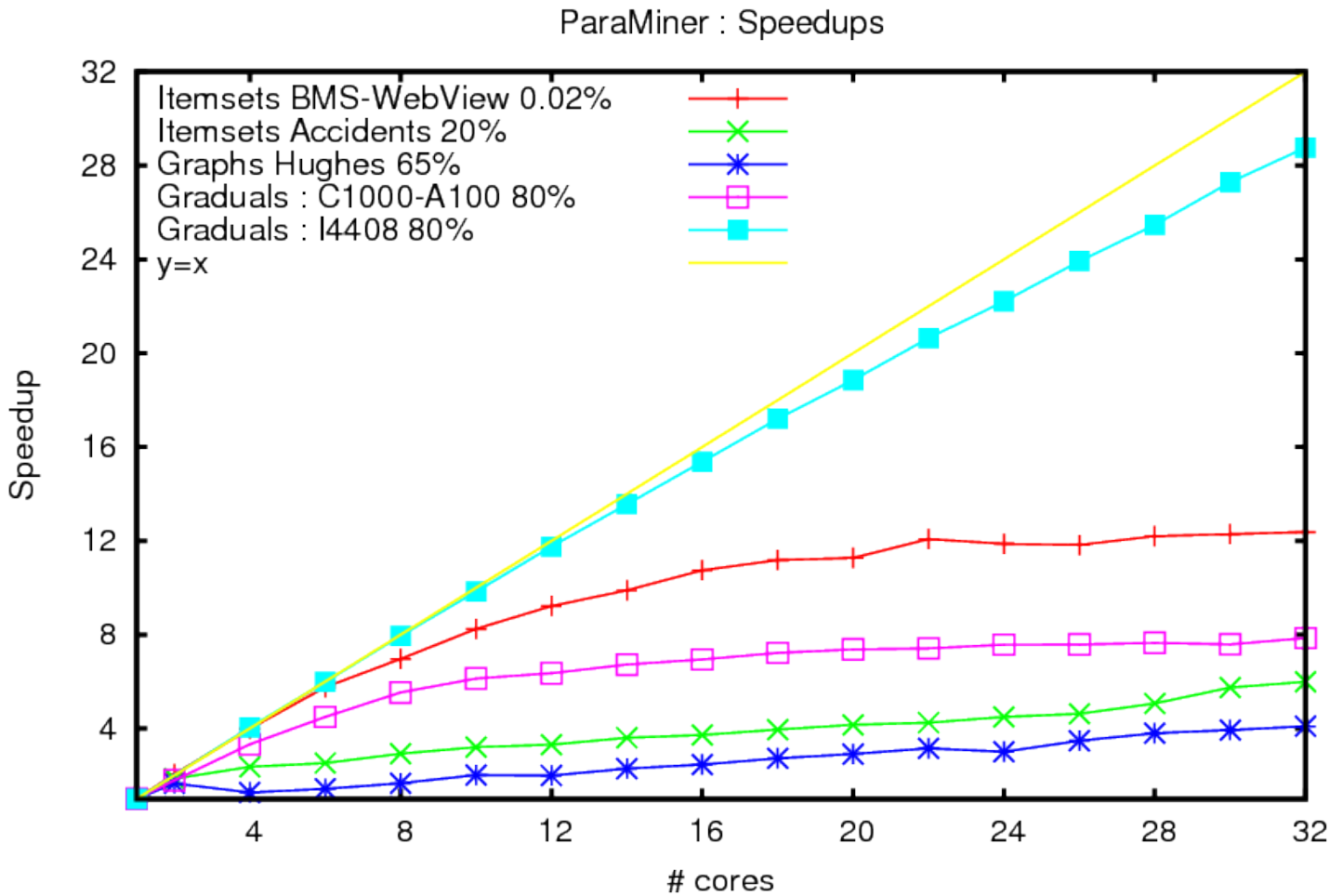
Speedup measurements

1. Laptop

- speedup on 4 cores: 3 to 4

Speedup measurements

2. Server



Performance analysis

In [Buerhrer 2006, Tatikonda 2008], authors exhibit two important issues:

- load imbalance
- memory issues

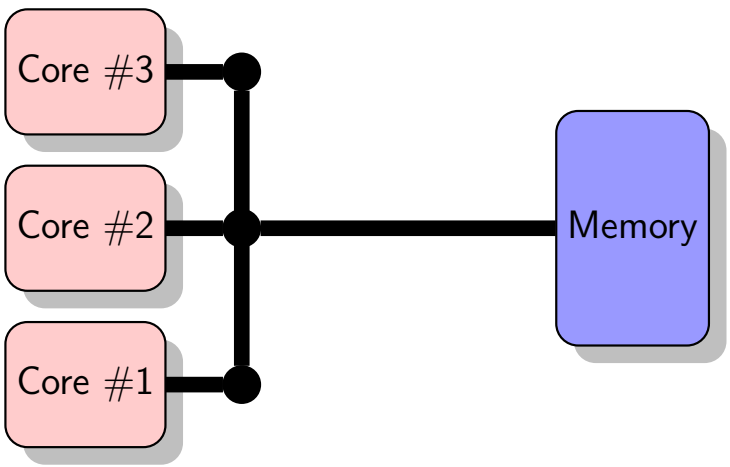
Performance analysis

In [Buerhrer 2006, Tatikonda 2008], authors exhibit two important issues:

- load imbalance
- memory issues

Bus contention

Memory bus: connect cores to memory



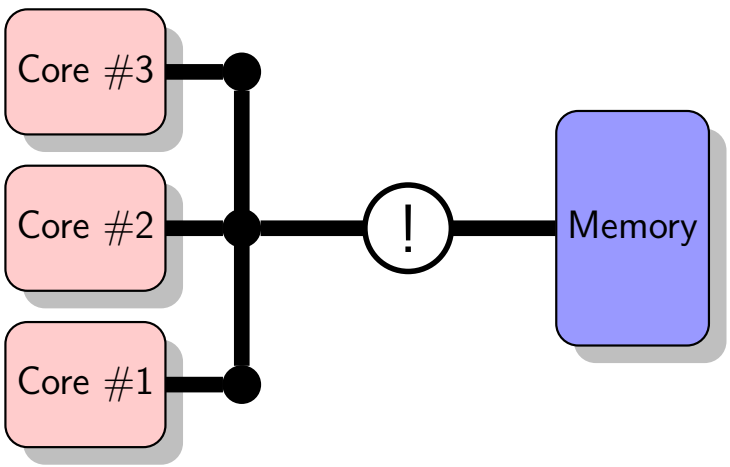
Performance analysis

In [Buerhrer 2006, Tatikonda 2008], authors exhibit two important issues:

- load imbalance
- memory issues

Bus contention

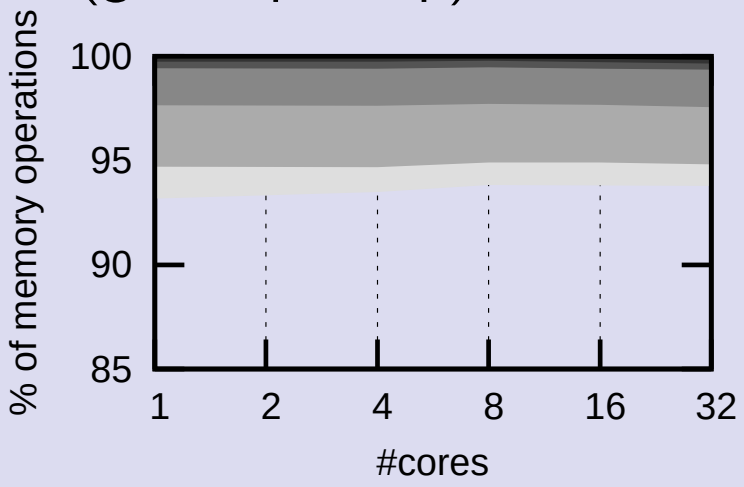
Memory bus: connect cores to memory
too many memory access ▶ subject to contention



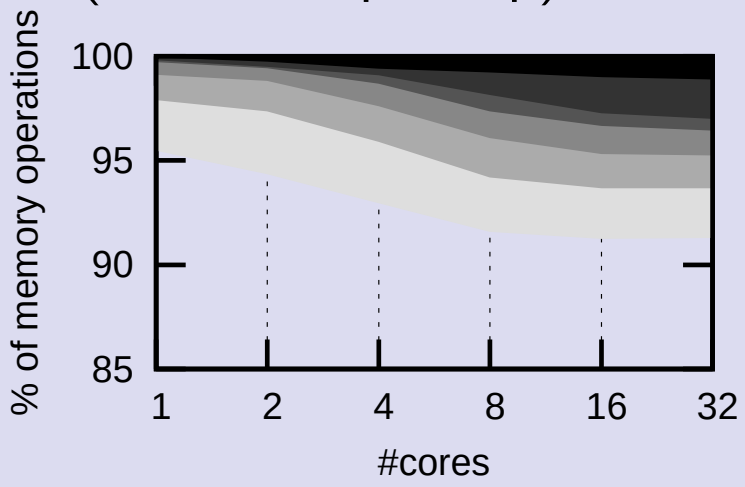
Bus contention in ParaMiner

■ % hi-lat. memory operation VS # of cores

gradual itemset mining
(good speedup)



frequent itemset mining
(bounded speedup)



dark slices = high latencies memory operations

- **GRI**: more cores ▶ % hi-lat. memory operations remains constant
- **FIM**: more cores ▶ increasing % of hi-lat. memory operations

Melinda's approach

[Tatikonda 2008] proposed **architecture conscious** pattern mining algorithm

Requires deep modifications in the algorithm

Inadequate to ParaMiner:

- could penalize executions that perform well

Melinda's *tuple distribution strategies*

- distribute tuples to available cores
- user-defined
  - can exploit **algorithmic level** information (in tuples)
  - can exploit **architectural level** information (available in Melinda)

Tuple distribution strategies

Structured tuplespace

Tuple space is divided into **internals**

- Internals can be used to classify tuples in the tuple space

Defining new strategies

- *distribute*() defines in which internal to put the tuple
- *retrieve*() from which internal the tuple has to be taken

- supports heavy usage

Architecture conscious tuple distribution strategy

1. **Distribute**: Tuples sharing datastructures go in the same internal
2. **Retrieve**: Cores of the same processor retrieve tuples from the same internal

Architecture conscious tuple distribution strategy

1. **Distribute**: Tuples sharing datastructures go in the same internal
2. **Retrieve**: Cores of the same processor retrieve tuples from the same internal

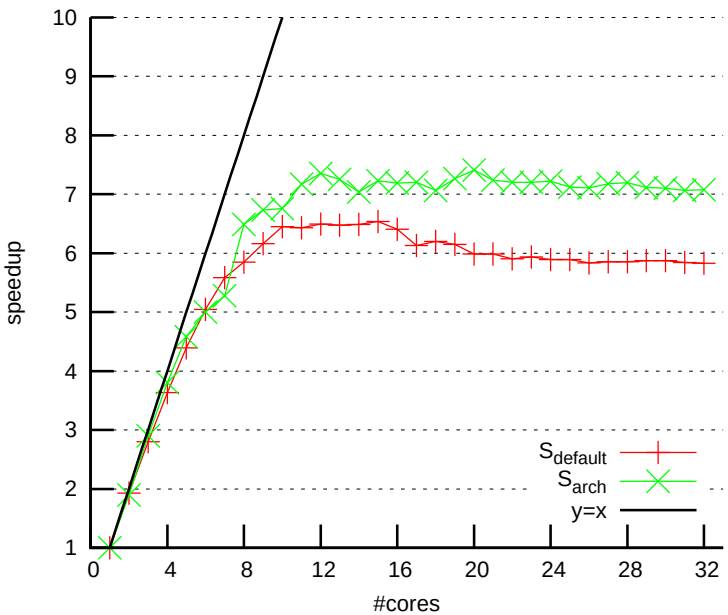
- **Improve cache usage** and **reduce bus contention**

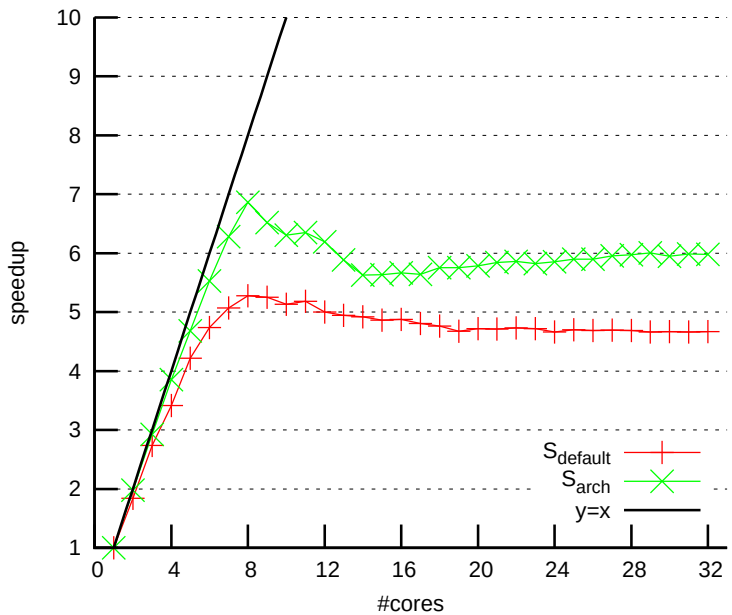
- **Frequent itemset mining, BMS-WebView/Accidents dataset**

dataset	# items	# transactions	dataset size	Density (%)
BMS-WebView-2	3,340	77,512	320,601	0.14
Accidents	468	340,184	11,500,870	7.22

Architecture conscious tuple distribution strategy

1. **Distribute**: Tuples sharing datastructures go in the same internal
2. **Retrieve**: Cores of the same processor retrieve tuples from the same internal





No source code modification ▶ 30% performance gain

Outline

Comparative experiments: frequent itemset mining

Algorithms

- **PLCM** [Negrevergne 2010] parallel implementation of LCM [Uno 2004] FIMI'04 Award
- **MT-Closed** [Lucchese 2007] parallel implementation of DCI-Closed [Lucchese 2004]

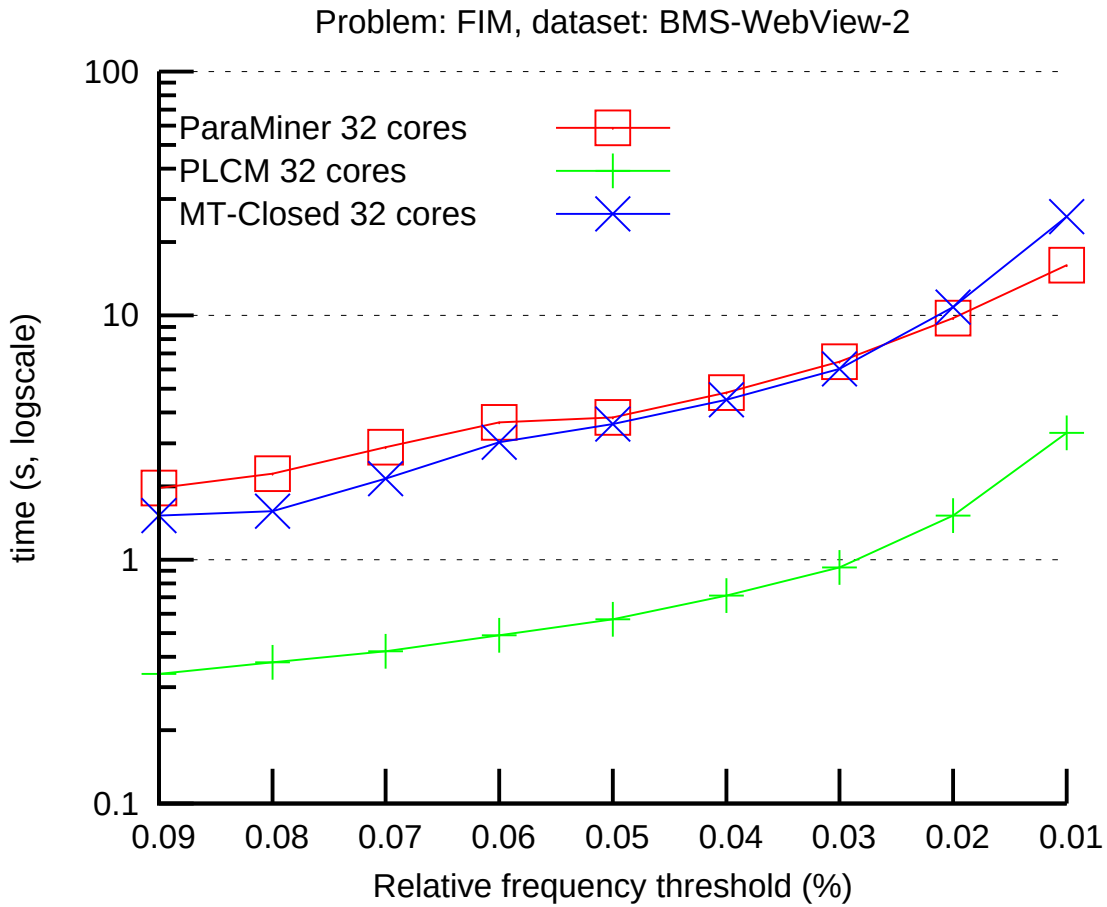
Datasets

dataset	# items	# transactions	dataset size	Density (%)
BMS-WebView-2	3,340	77,512	320,601	0.14
Accidents	468	340,184	11,500,870	7.22

$$density(\mathcal{D}_E) = \frac{\#items \times \#transactions}{|\mathcal{D}_E|}(\times 100)$$

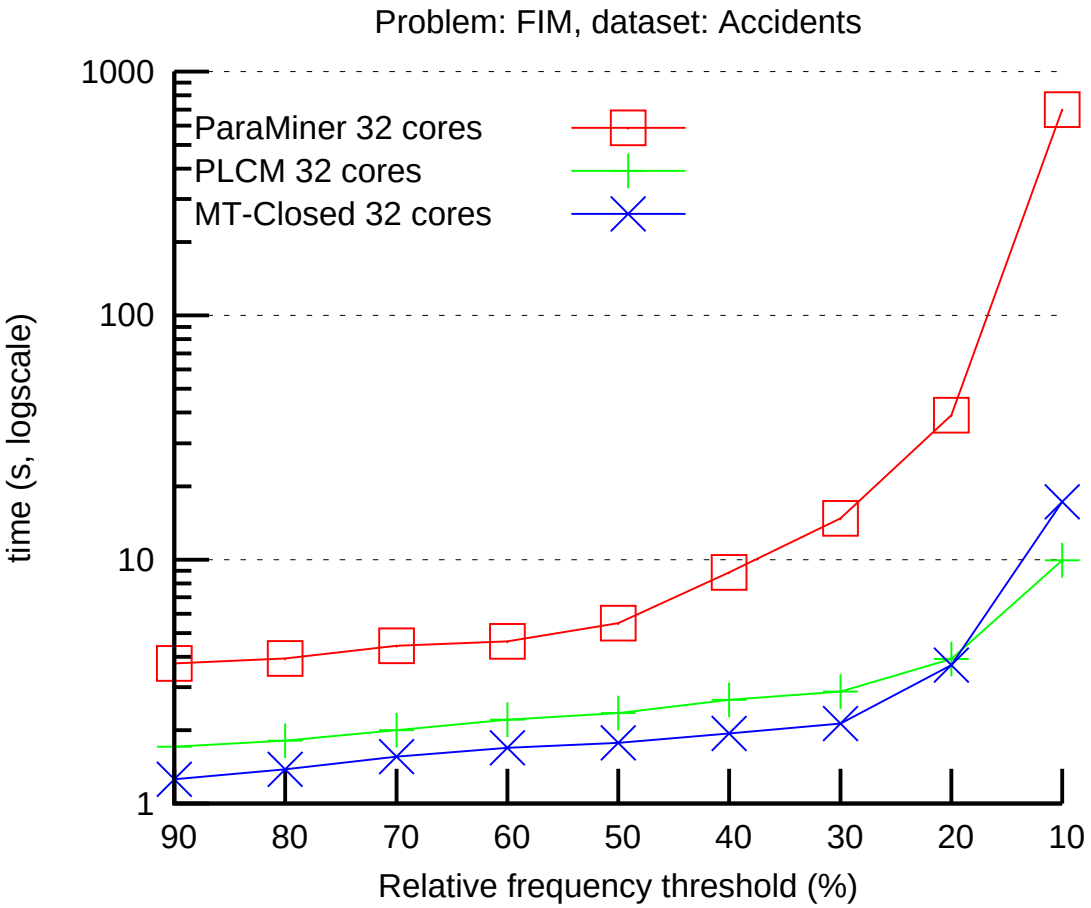
Comparative experiments frequent itemset mining (2)

BMS-WebView-2 (sparse)



Comparative experiments frequent itemset mining (2)

Accidents (dense)



Comparative experiments: gradual itemset mining

Algorithms

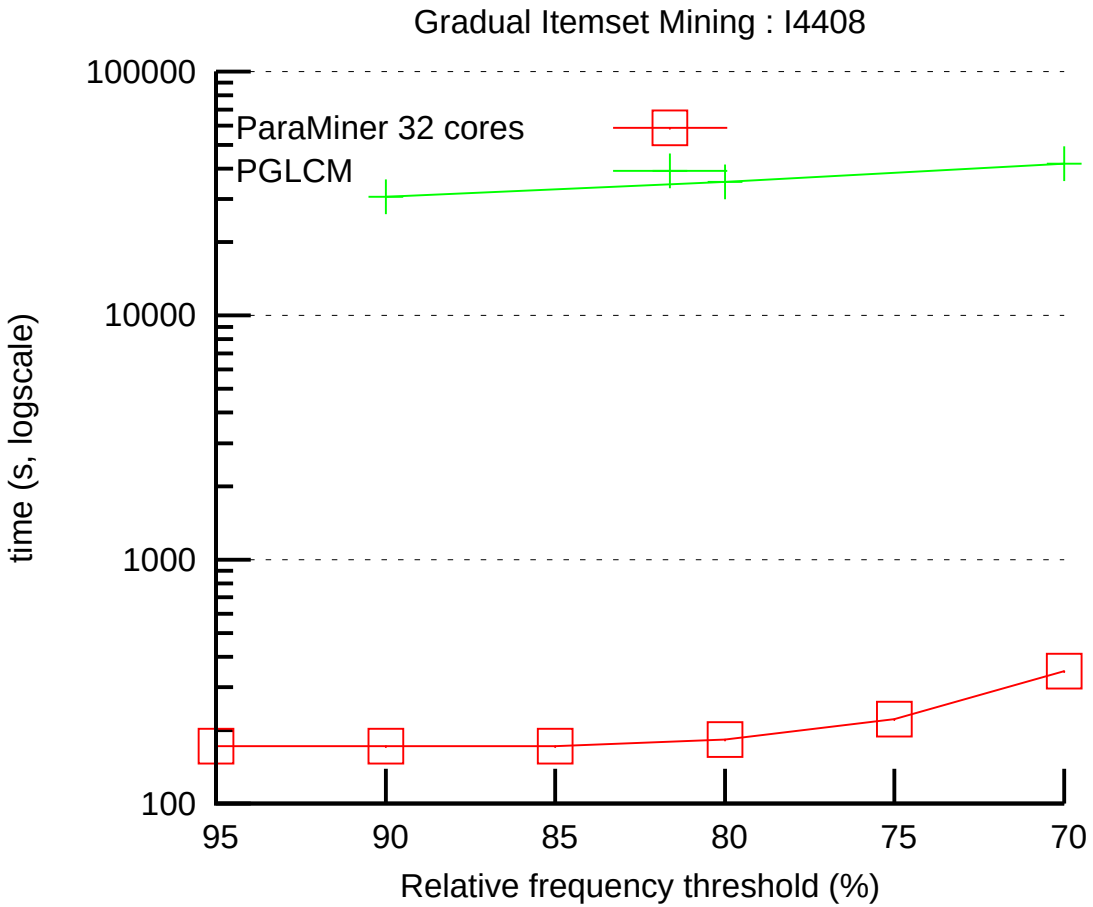
- **PGLCM** [Do 2010]
- **PGP-mc** (non-closed) [Laurent 2010]

Datasets

dataset name	ground set size	# transactions	dataset size	Density (%)
I4408	8824	11,556	50,985,072	50

Comparative experiments gradual itemset mining (2)

I4408 (Real)



Outline

Generic framework and problem statement

ParaMiner
- Efficient exploration of the set of candidate patterns
- Speeding up candidate pattern testing
- Parallel execution of ParaMiner

Experiments
- Parallel performance evaluation
- Comparative experiments

Conclusion and future work

Conclusion

- **ParaMiner**
  - based on state of the art on pattern enumeration
    - ▶ useful to parallel pattern mining
  - designed an efficient generic dataset reduction
  - exploit multi-core architectures
- **Melinda**
  - parallel engine of ParaMiner (and other algorithms)
  - extensible through strategies

Conclusion

- **ParaMiner**
  - based on state of the art on pattern enumeration
    - ▶ useful to parallel pattern mining
  - designed an efficient generic dataset reduction
  - exploit multi-core architectures
- **Melinda**
  - parallel engine of ParaMiner (and other algorithms)
  - extensible through strategies

Novelty

- ParaMiner: an efficient solution for new pattern mining problems

  New pattern mining problem can benefit from 15 years of research
- ParaMiner/Melinda: tool to learn about parallel pattern mining

Future works

Extend ParaMiner's genericity

- Generalize strong accessibility properties to other structures
  ▶ handle sequences, and general graphs
- Study partial strong accessibility of set of patterns
  ▶ bridge with constraint pattern mining

Extend ParaMiner/Melinda's efficiency

- Improve Melinda's strategies ▶ More flexible and expressive querying tuples
- Target larger computing platforms ▶ cluster of computers

Thank you !