The Difference and the Norm

Kailash Budhathoki & **Jilles Vreeken**

UNIVERSITÄT DES SAARLANDES
CLUSTER OF EXCELLENCE
max planck institut informatik

Question of the Day

Say, we have **more than one** database over the same domain

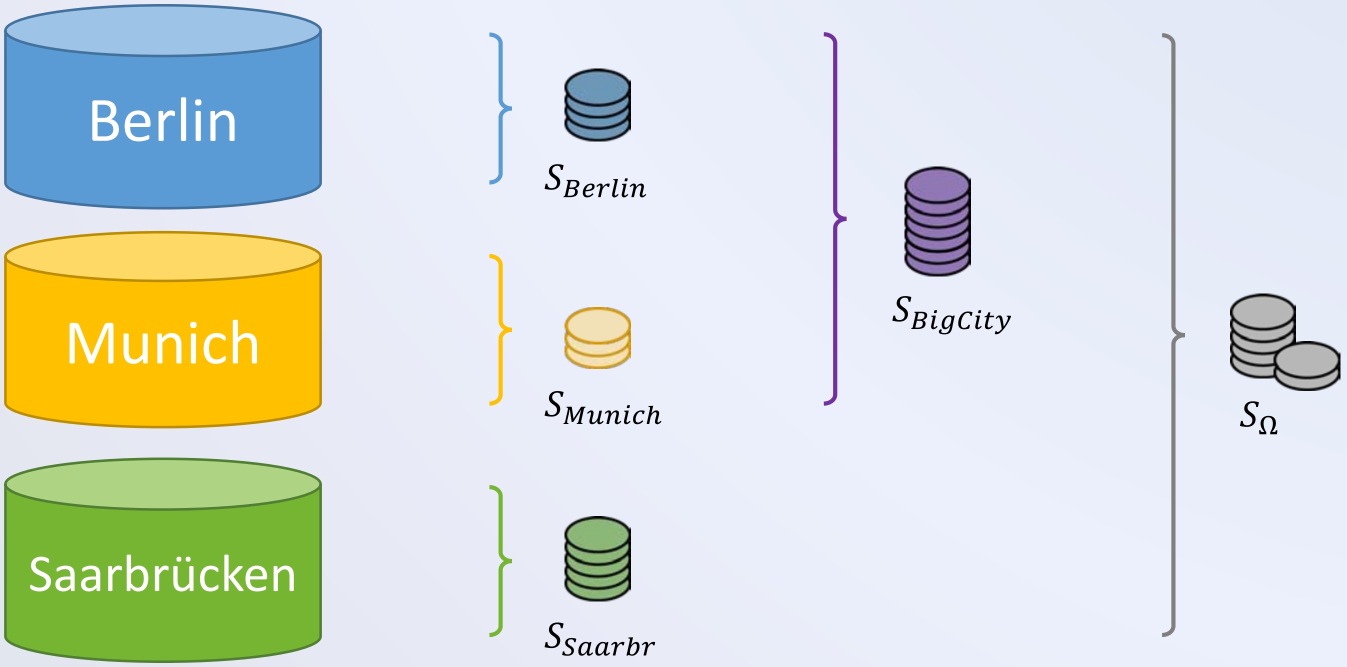
How can we characterise the **similarities** -and- **differences** between these databases?

How can we do this **without redundancy**, and **without setting parameters**?

What we want, informally

What we want, informally

A **global** model $\mathcal{S}$ consisting of pattern sets $S \in \mathcal{S}$, that give **local detail** and **together** are optimal for $\mathcal{D}$.

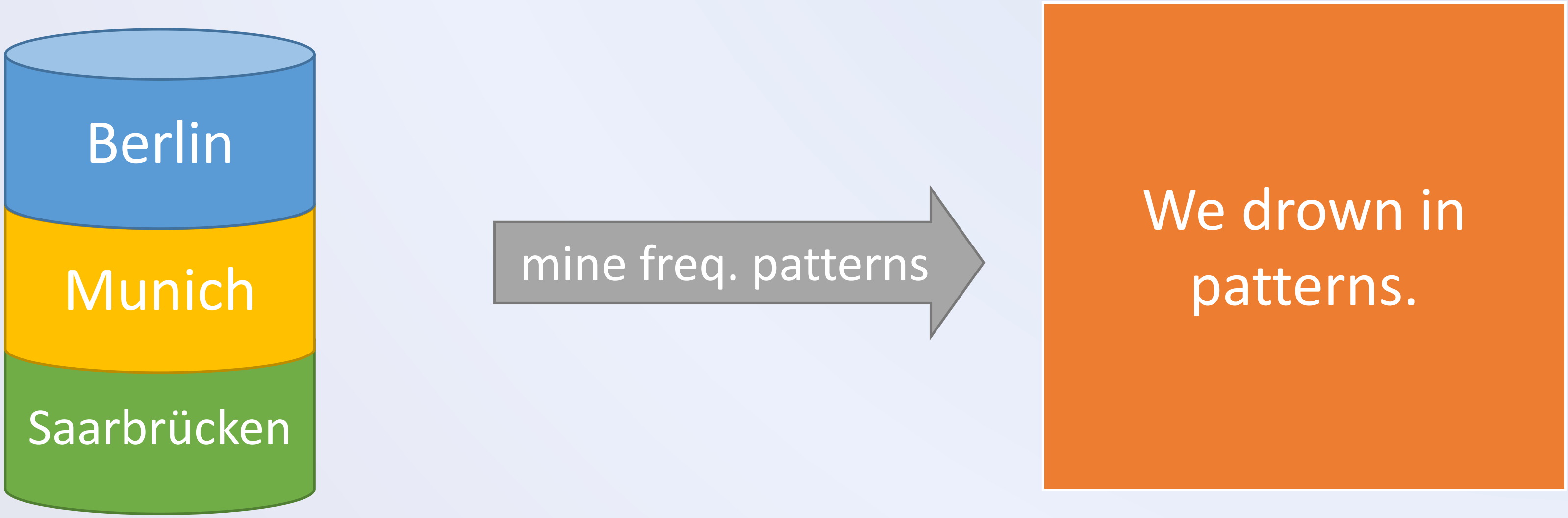


The Traditional Approach

We run a chain of supermarkets.
We have **one** database.

The Traditional Approach

We run a chain of supermarkets.
We have **one** database.



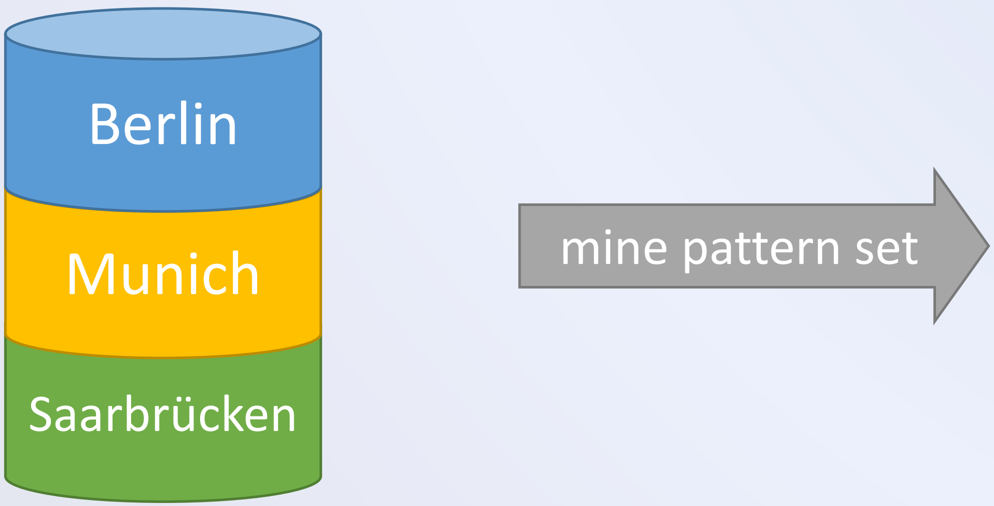
The Available Approach

We run a chain of supermarkets.
We have **one** database.

for example, using KRIMP, or SLIM

The Available Approach

We run a chain of supermarkets.
We have **one** database.

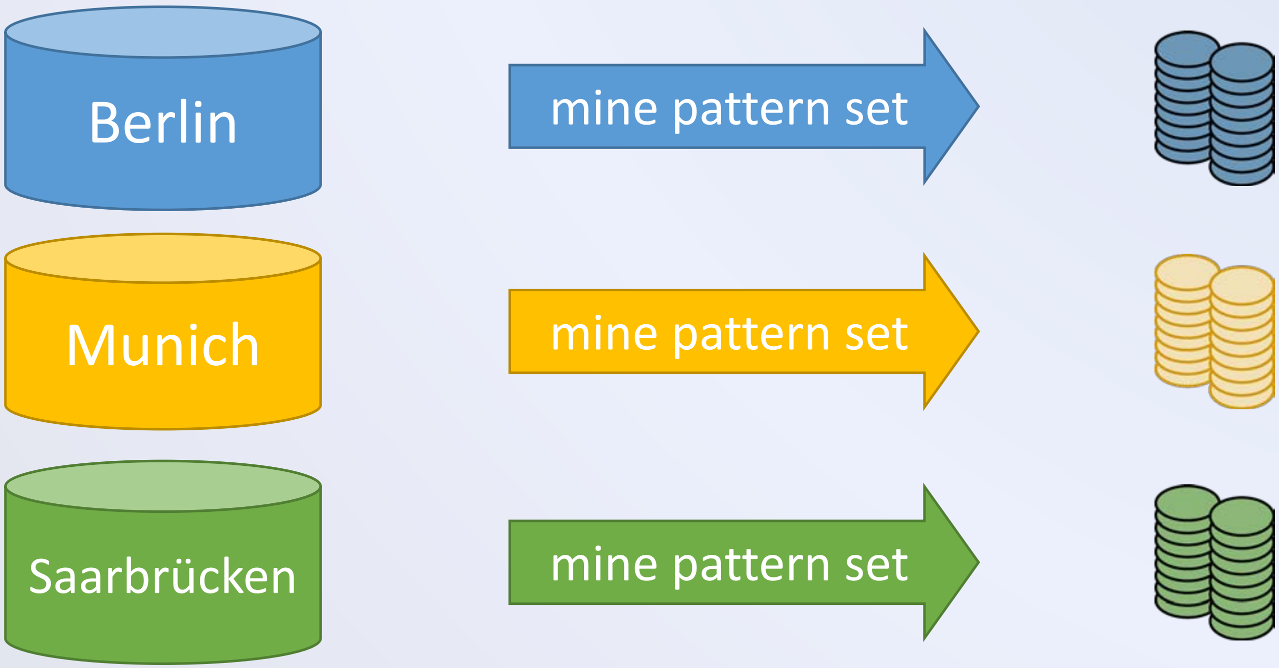


We only get a **global** overview, **not** what's most important per store!

for example, using KRIMP, or SLIM

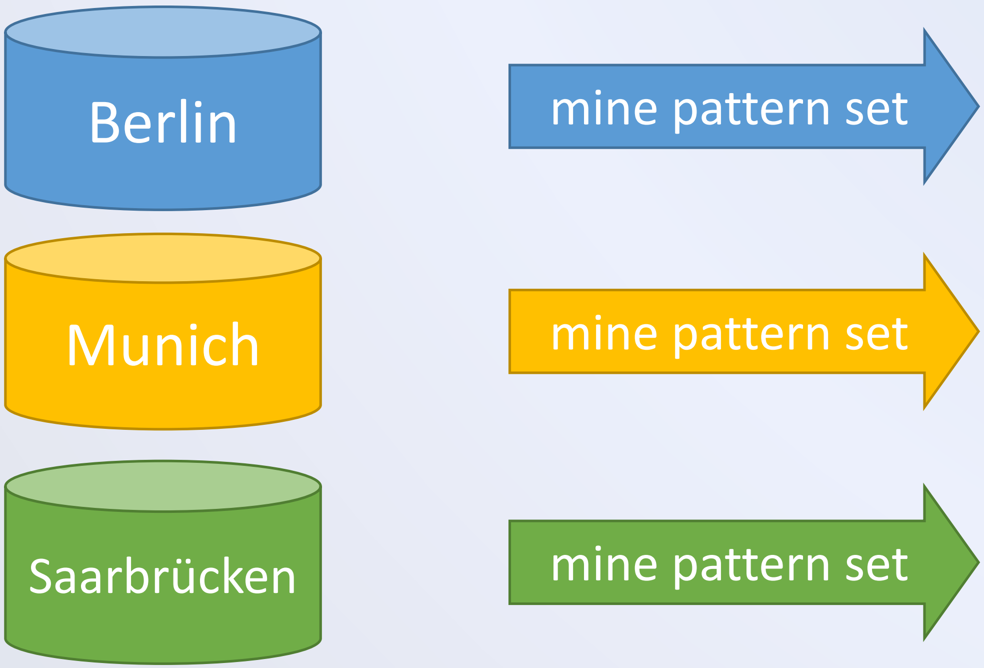
The Available Approach

We run a chain of supermarkets.
We have **multiple** databases.



The Available Approach

We run a chain of supermarkets.
We have **multiple** databases.

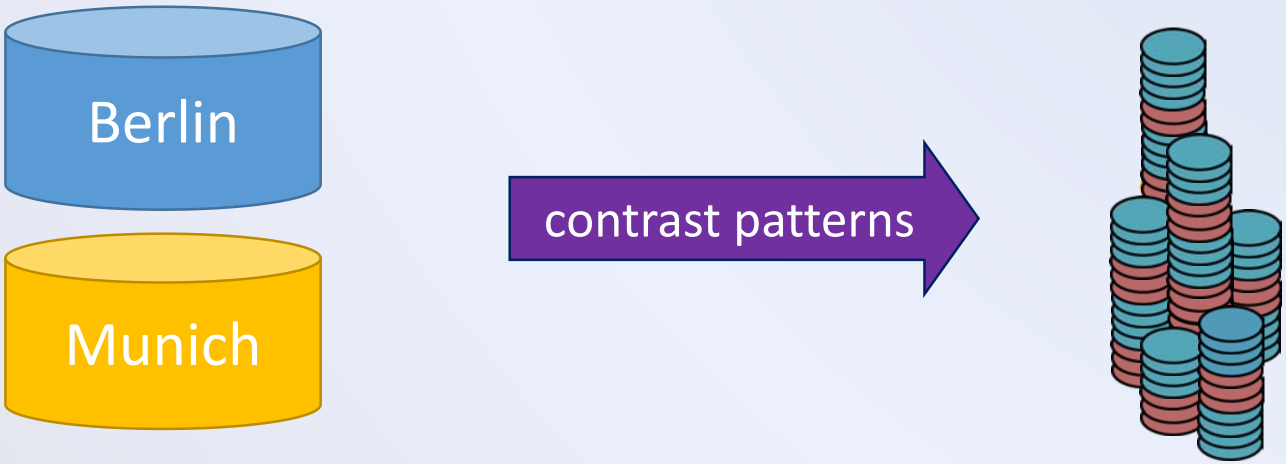


Now we have only **local** detail.

Between these many patterns will be **redundant** yet **subtly different**

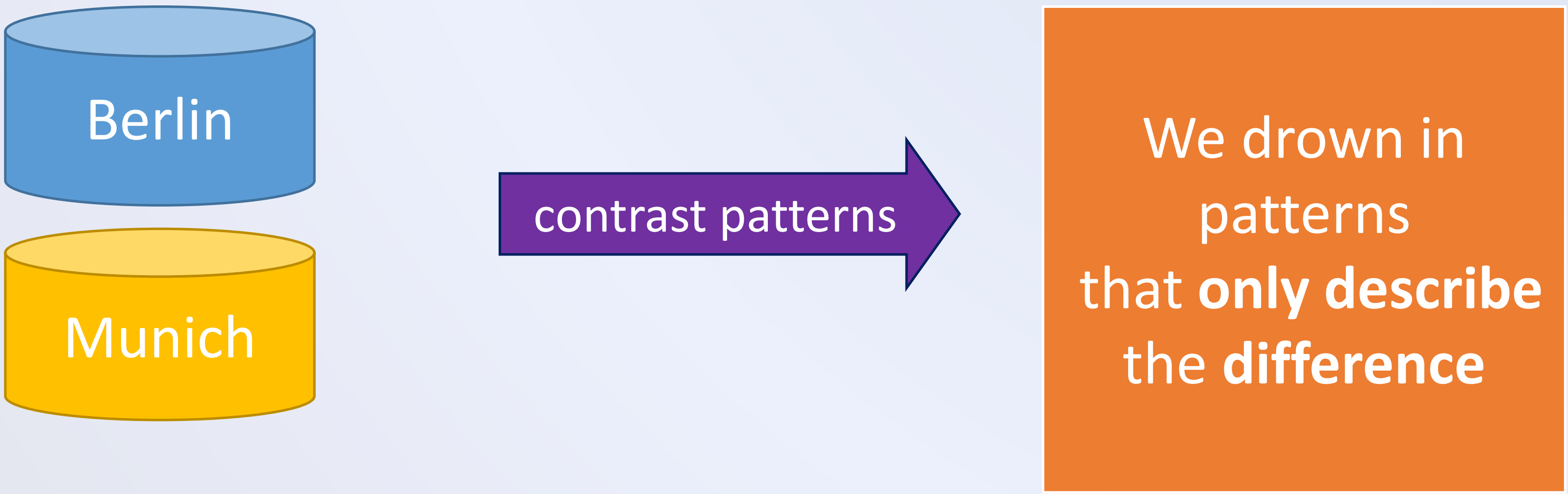
The Available Approach

We run a chain of supermarkets.
We have **two** databases.



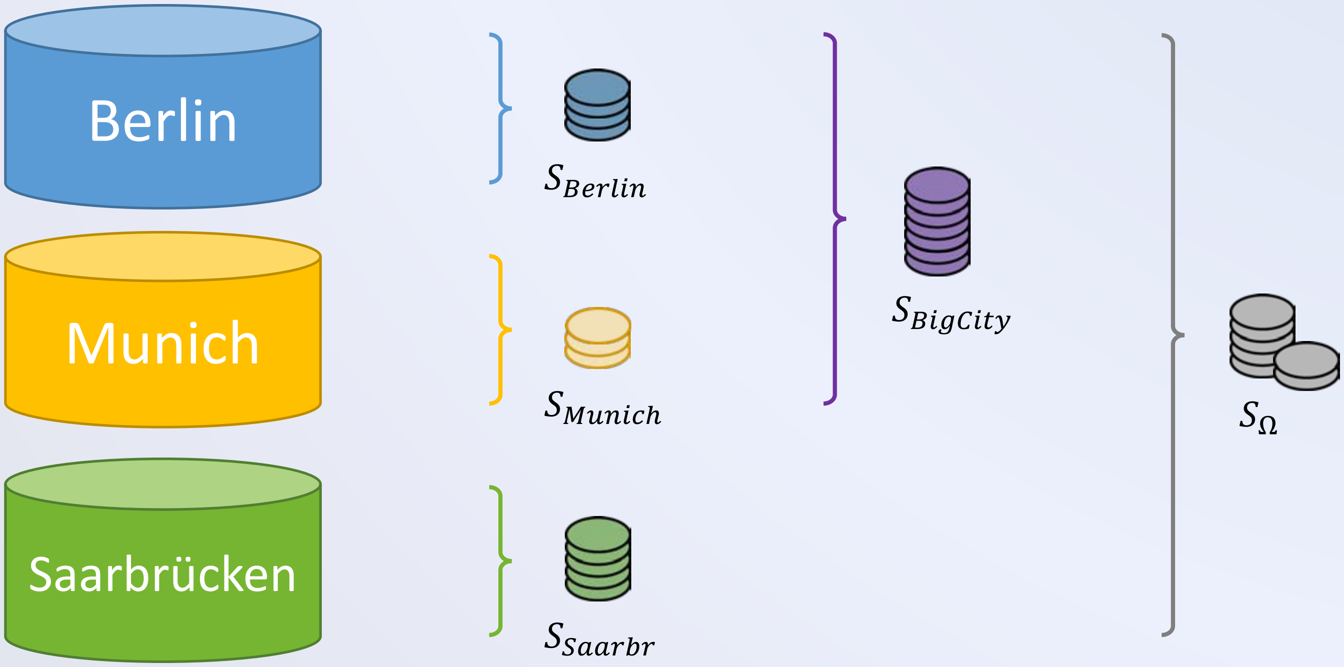
The Available Approach

We run a chain of supermarkets.
We have **two** databases.



What we want, informally

A **global** model $\mathcal{S}$ with **local** detail $S_i \in \mathcal{S}$,
without redundancy.



What we want, formally

Let $\mathcal{I}$ be a set of items.

Let $\mathcal{D}$ be a bag of transaction databases $D_i \in \mathcal{D}$ over $\mathcal{I}$.

Let U be a set of index sets over $\mathcal{D}$, with every $j \in U$ identifying a subset of $\mathcal{D}$ the user wants to be characterised.

Discover the **set $\mathcal{S}$ of pattern sets**

where each pattern set $S_j \subseteq \mathcal{P}(\mathcal{I})$, and

such that there is a pattern set $S_j \in \mathcal{S}$ for every $j \in U$,

that **best characterises $\mathcal{D}$**

MDL

The Minimum Description Length (MDL) principle

given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$
is that M that minimises

$$L(M) + L(D \mid M)$$

in which

$L(M)$ is the length, in bits, of the description of M

$L(M \mid D)$ is the length, in bits, of the description of the data when encoded using M

(see, e.g., Rissanen 1978, Grünwald, 2007)

What we want, formally

Let $\mathcal{I}$ be a set of items, $\mathcal{D}$ a bag of transaction databases $D_i \in \mathcal{D}$ over $\mathcal{I}$, and U a set of index sets over $\mathcal{D}$, with every $J \in U$ identifying a subset of $\mathcal{D}$ the user wants to be characterised.

Discover the set $\mathcal{S}$ of pattern sets for which

$$L(\mathcal{S}) + L(\mathcal{D} \mid \mathcal{S})$$

is minimal.

Note: patterns will **only** be included if they aid to describe the data more succinctly, and then only in as few as necessary pattern sets

Describing the Data

We know what our models are, let's discuss how we describe the data

$$L(\mathcal{D} \mid \mathcal{S}) = \sum_{D_i} L(D_i \mid C_i)$$

We describe D_i using only the pattern sets in $\mathcal{S}$ that are relevant for D_i. For example, only S_{Saarbr} and S_Ω are relevant for D_{Saarbr}.

To ensure any transaction can be encoded, we always add all singletons.

Together, we call these the coding set C_i for database D_i

(see, e.g., Rissanen 1978, Grünwald, 2007)

Coding set C

	Itemset	Usage
S_{Ω}	A C	0
S_{Saarbr}	B D	0
	C E	0
	A	0
	B	0
	C	0
	D	0
	E	0

Transaction t

B C E

(Similar to Siebes et al 2006, Vreeken et al. 2011)

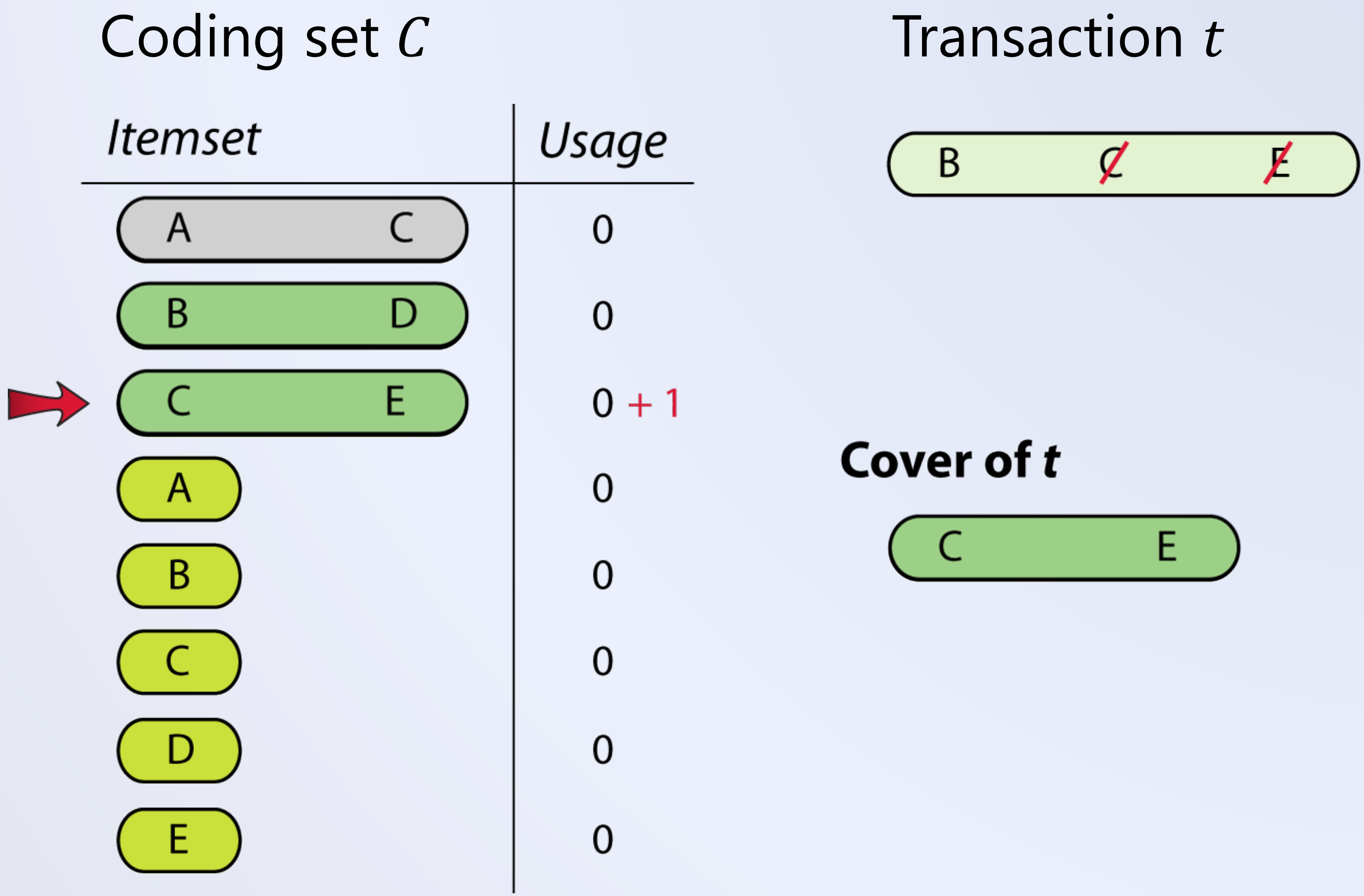
Coding set C

Itemset	*Usage*
A C	0
B D	0
C E	0
A	0
B	0
C	0
D	0
E	0

Transaction t

B C E

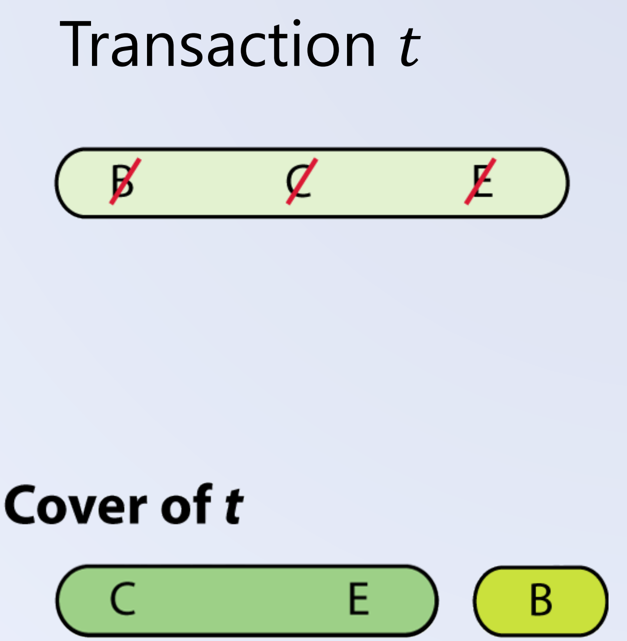
(Similar to Siebes et al 2006, Vreeken et al. 2011)



(Similar to Siebes et al 2006, Vreeken et al. 2011)

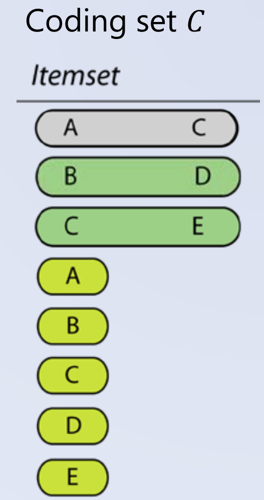
Coding set C

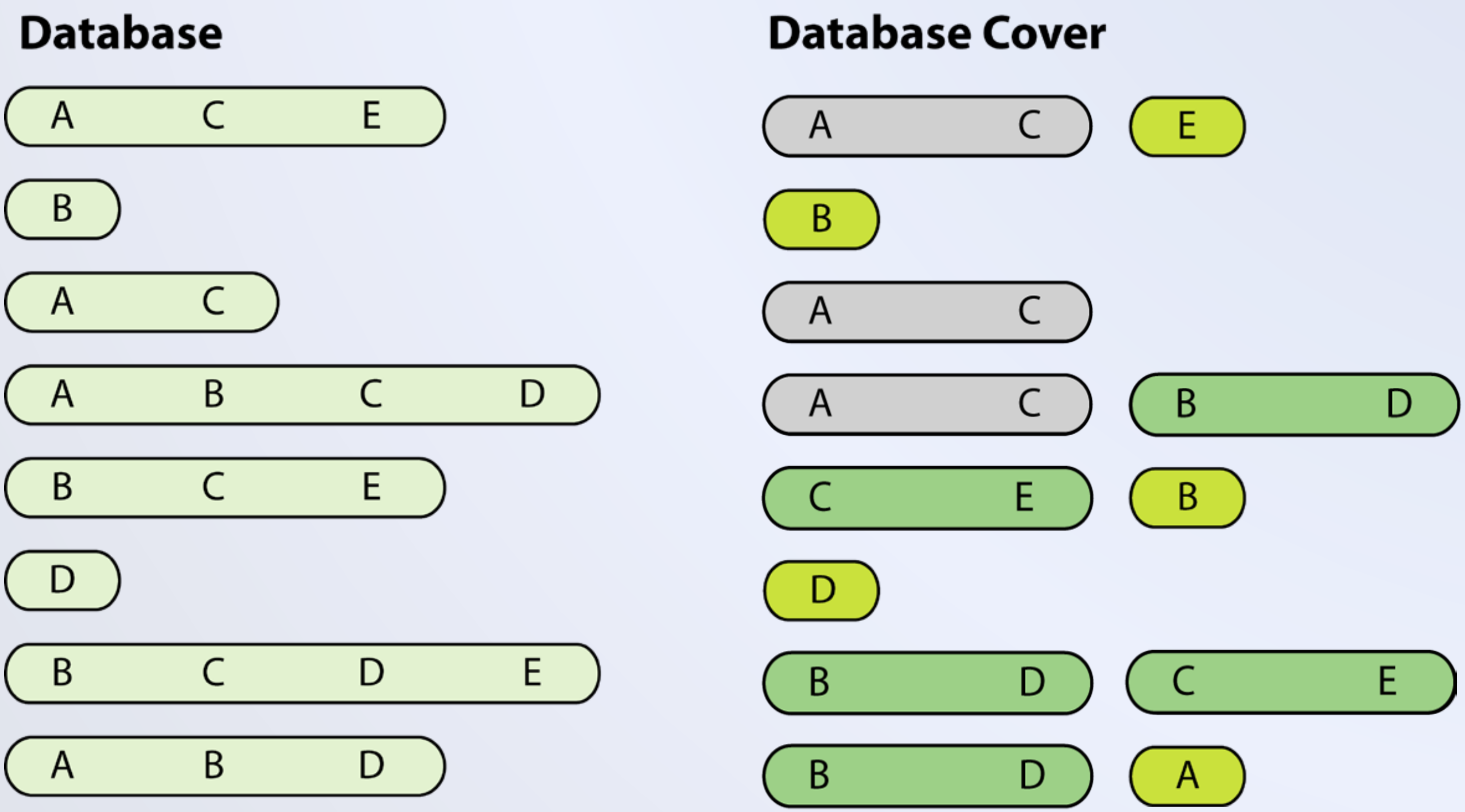
Itemset	*Usage*
A C	0
B D	0
C E	1
A	0
→ B	0 + 1
C	0
D	0
E	0



(Similar to Siebes et al 2006, Vreeken et al. 2011)

Encoding a database





(Similar to Siebes et al 2006, Vreeken et al. 2011)

Optimal prefix codes

The probability for $X \in C$ in the cover of D is

$$P(X \mid C, D) = \frac{usage(X)}{\sum_{Y \in CT} usage(Y)}$$

The optimal code for the coding distribution P assigns a code to $X \in C$ with length

$$L(X \mid C, D) = -\log(P(X \mid C, D))$$

(Shannon, 1948; Thomas & Cover, 1991)

A simple life

To encode optimally, we require actual usages.
Assuming these, the encoded size of a database is

$$L(D_i \mid C_i) = \sum_{X \in C_i} usage(X) L(X \mid C_i, D_i)$$

However… we **do not want** to know the usages!

Do not want

When we

- store usages **per database**
  we **encode optimally** but **cannot reward generalisation**
  patterns are globally as expensive as they are locally

- store usages **per pattern set**
  we **reward generalisation** but with a **strong bias**
  to patterns with similar frequencies between databases

Hmm…

Prequential Coding

Can we encode optimally without knowing the usages?

Yes! By using **prequential coding**.

The idea is very simple

1) initialise all pattern usages to ϵ
2) send next code, increment its usage, repeat

This is order-invariant, rapidly approaches the true distribution, and within a constant factor of optimal!

(Grünwald 2007)

Prequential Coding

Can we encode **optimally** without knowing the usages?

Yes! By us

The idea

1) initiali

2) send

This is ord e

distribution, and within a constant factor of optimal!

By encoding **prequentially**
we can **reward patterns** that are
characteristic for multiple databases
beyond similar frequency!

(Grünwald 2007)

Prequential Coding, formally

Formally, things do get a bit more scary, as instead of

$$L(D \mid C) = \sum_{X \in C} usg(X) L(X \mid C)$$

we have to compute

$$L(D \mid C) = \log \Gamma(usg(C) + 0.5|C|) - \log \Gamma(0.5|C|) - \sum_{X \in \mathrm{C}} \left(\log\big((2usg(X) - 1)!!\big) - usg(X)\right)$$

Fortunately, both $\log \Gamma$ and $\log x!!$ can be approximated efficiently.

The Score

For conciseness, we skip the details on how to encode a model.

All that's left is to find that $\mathcal{S}$ that minimises

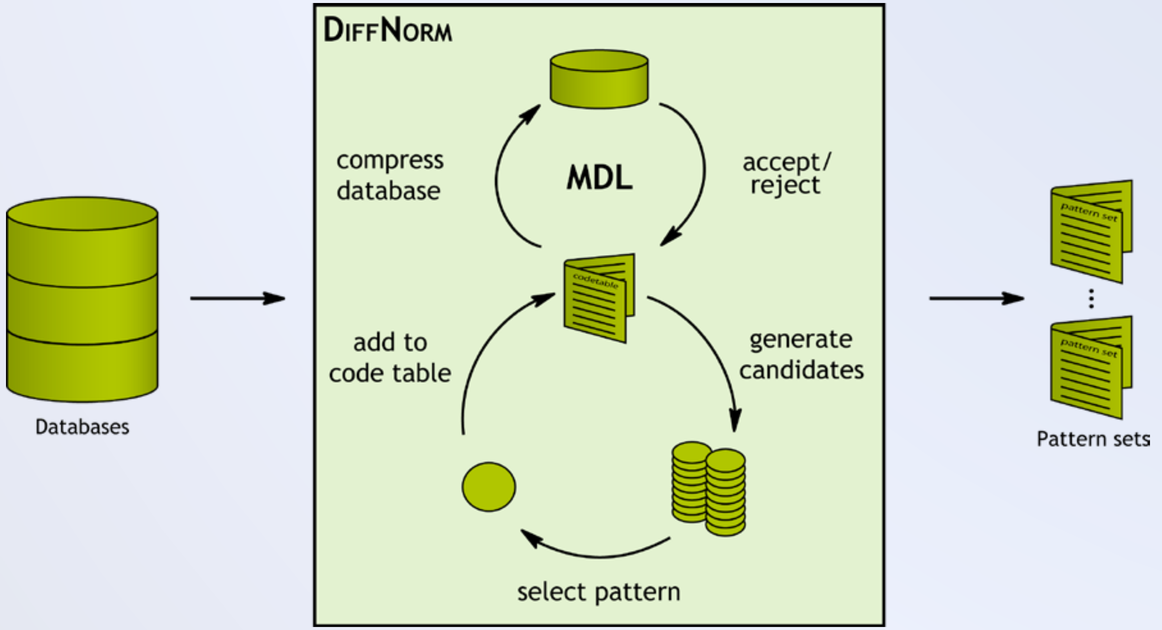
$$L(\mathcal{D}, \mathcal{S}) = L(\mathcal{S}) + L(\mathcal{D} \mid \mathcal{S})$$

This is easier said than done. The search space is enormous, the score is not convex, nor is it (anti-)monotonic.

Hence, we resort to heuristics.

The DiffNorm Algorithm

Main idea: iteratively reduce redundancy in the current description

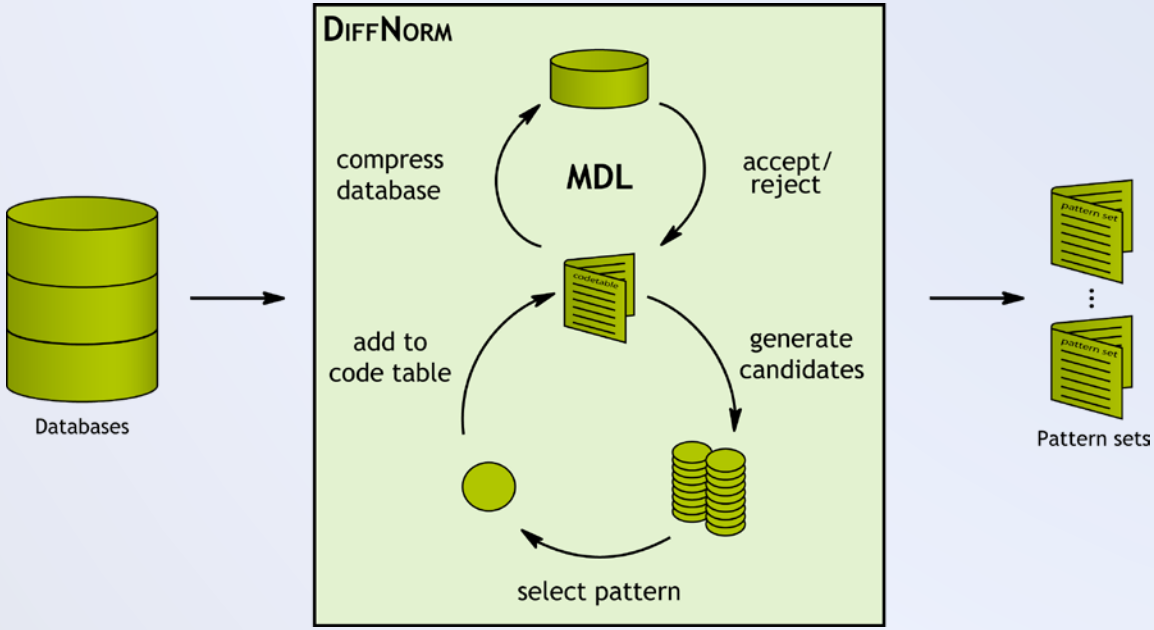


Evaluate each $X \cup Y$ of existing $X, Y \in \mathcal{S}$ for every coding set C_i

- determine its optimal assignment to $S_j \in \mathcal{S}$ s.t. compression is maximal

The DiffNorm Algorithm

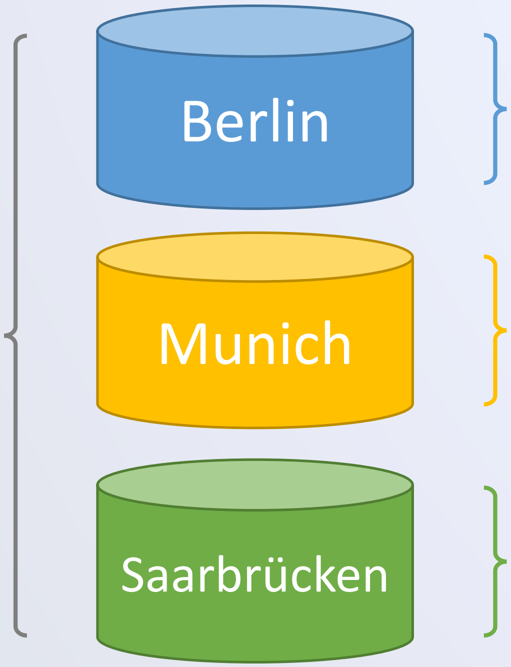
Main idea: iteratively reduce redundancy in the current description



Add that $X \cup Y$ to that subset of $\mathcal{S}$ s.t. compression is maximal

- re-consider every existing pattern, prune if it now harms compression

Refining the DiffNorm Algorithm



Evaluating **every pair** $X, Y \in \mathcal{S}$ is wasteful

- instead, we only consider X, Y that are co-used in the coding set C_i of any D_i

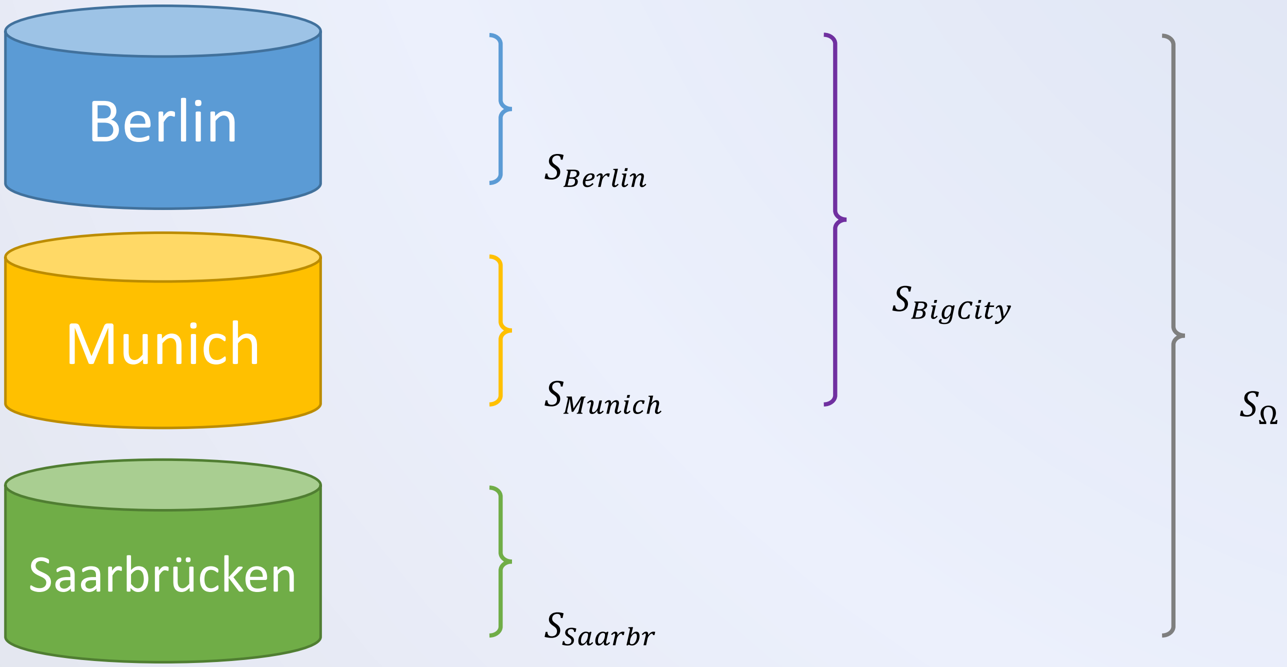
Evaluating **compression gain** is costly

- requires a full pass over the database
- instead, we **estimate** compression gain of a $X \cup Y$ based on the usages of X and Y

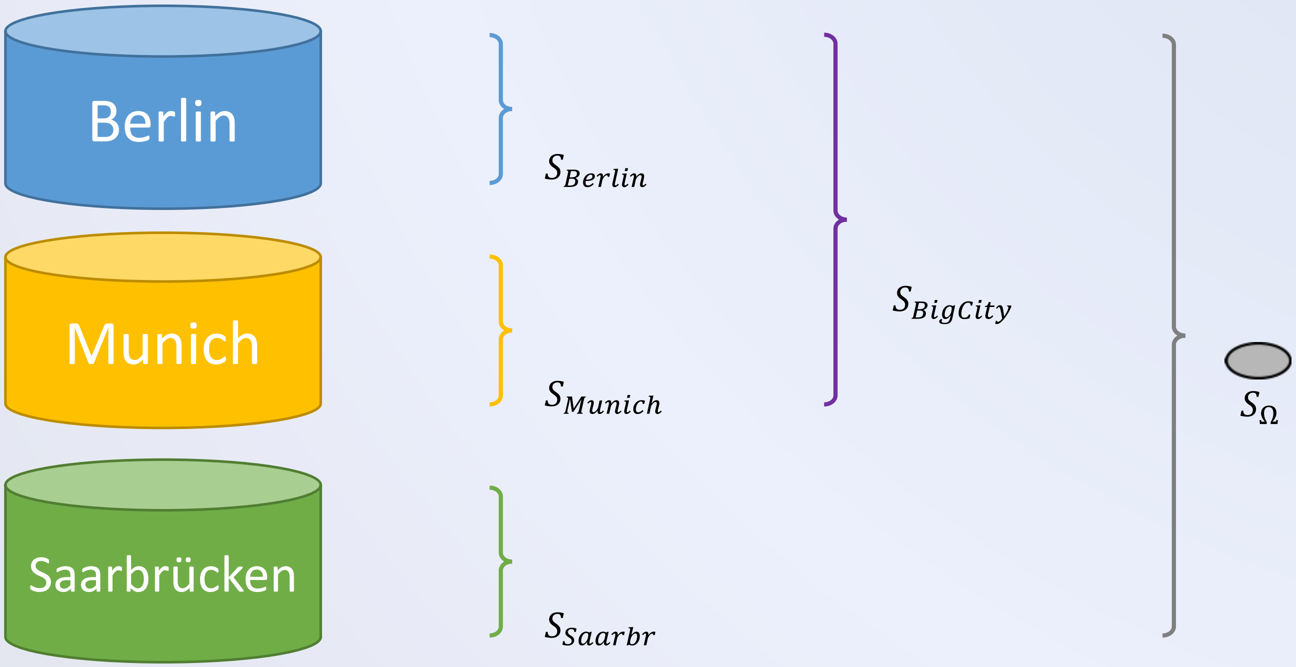
Finding the **true best candidate** is costly

- instead, we greedily consider in order of estimated gain; keep first with actual gain

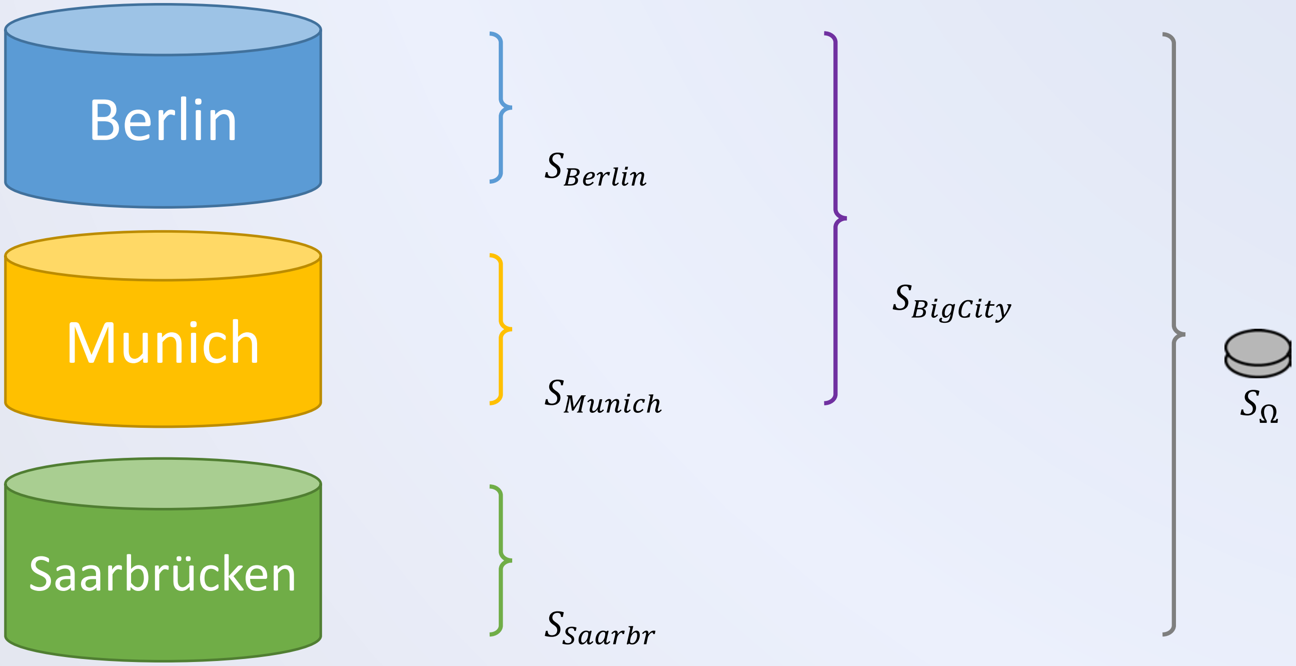
DIFFNORM in action (1)



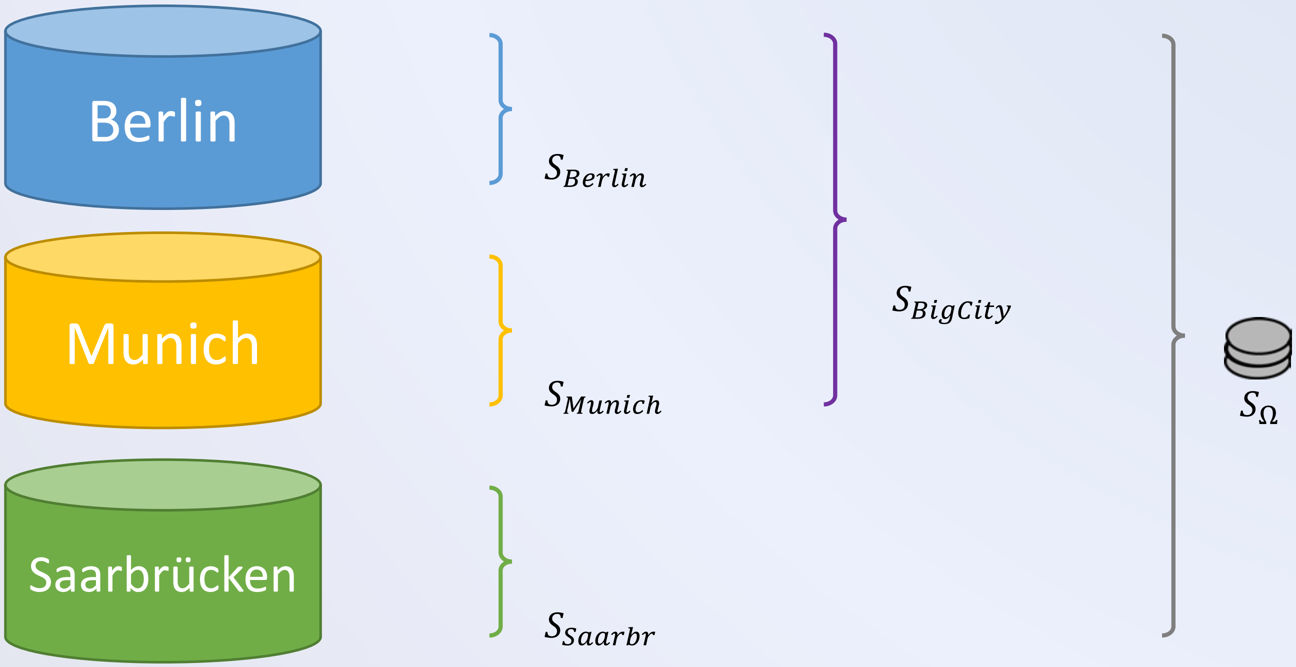
DIFFNORM in action (2)



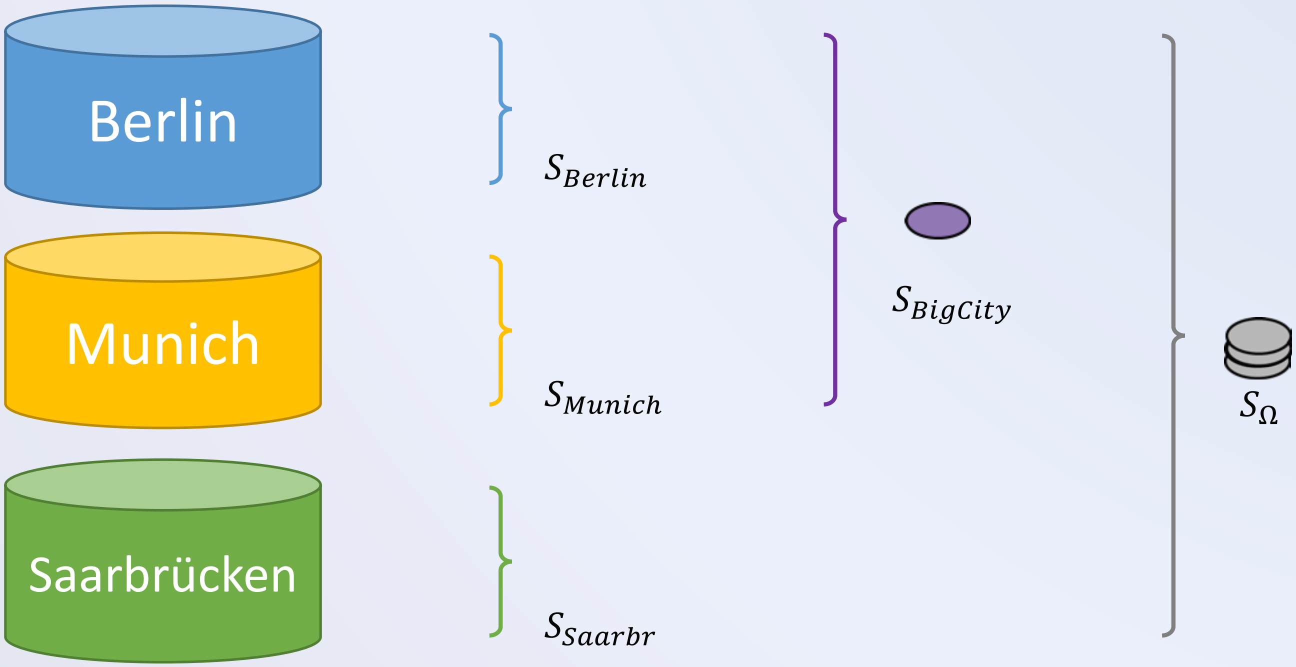
DIFFNORM in action (3)



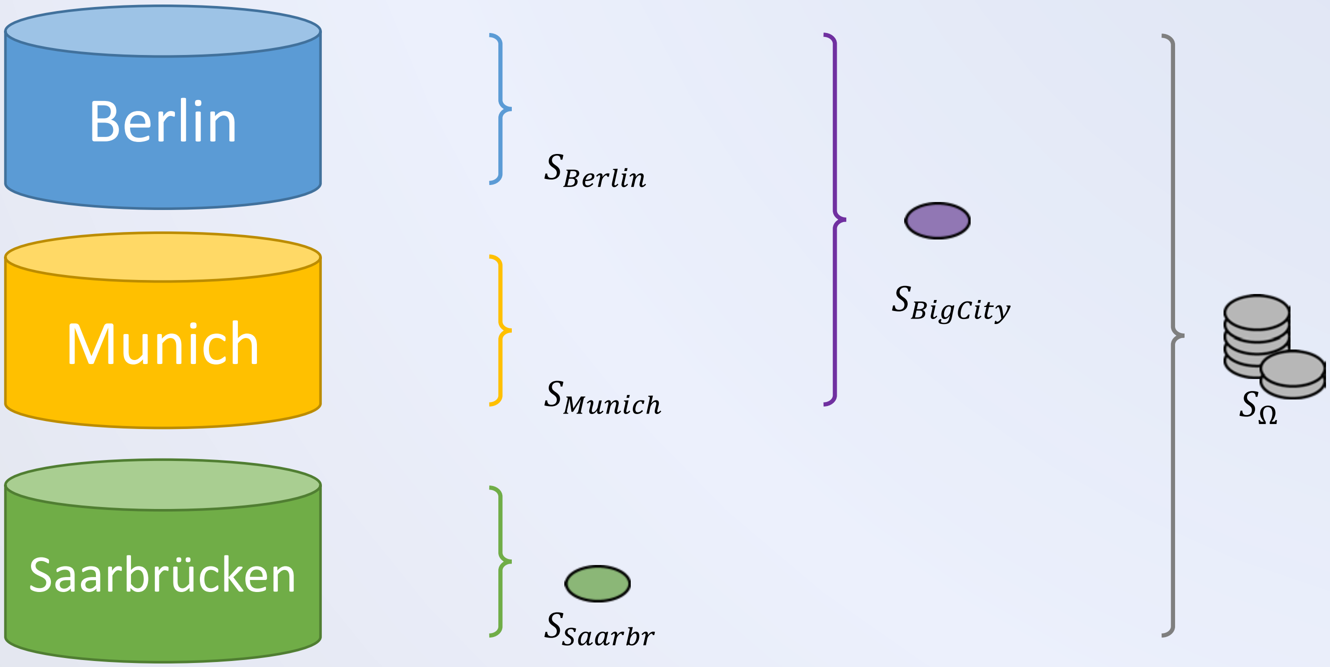
DiffNorm in action (4)



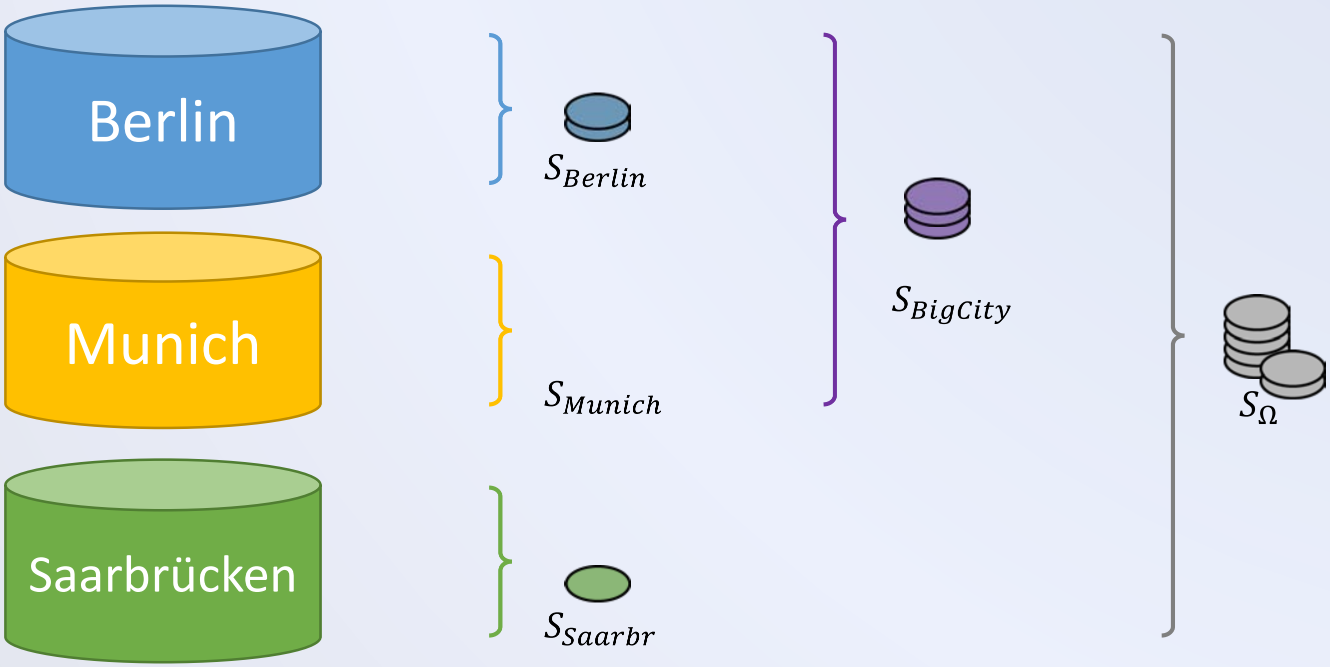
DIFFNORM in action (5)



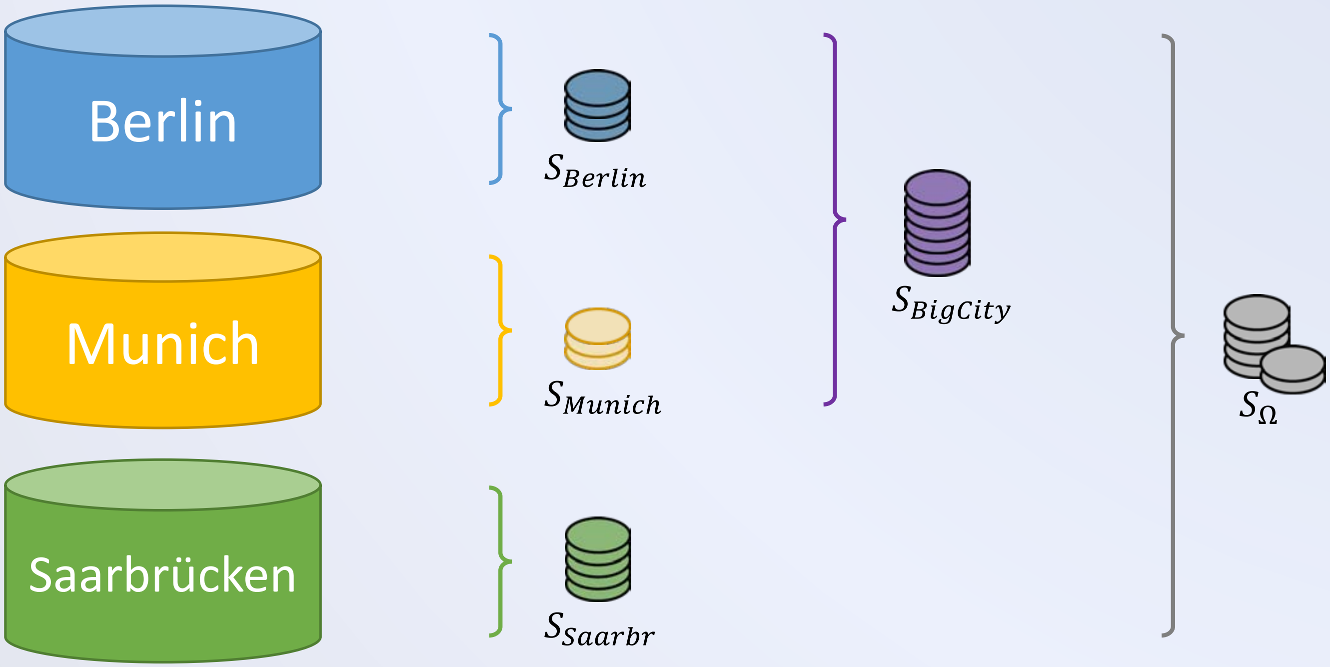
DiffNorm in action (6)



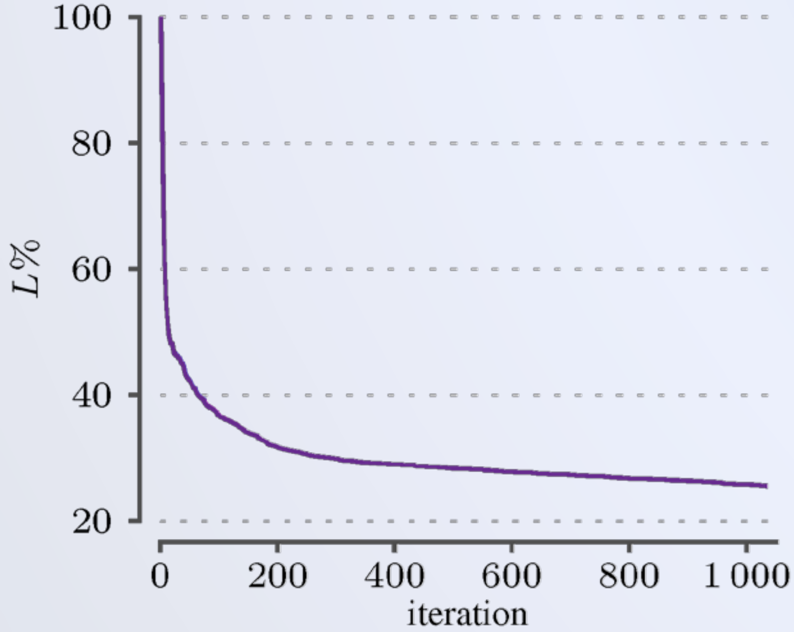
DiffNorm in action (7)



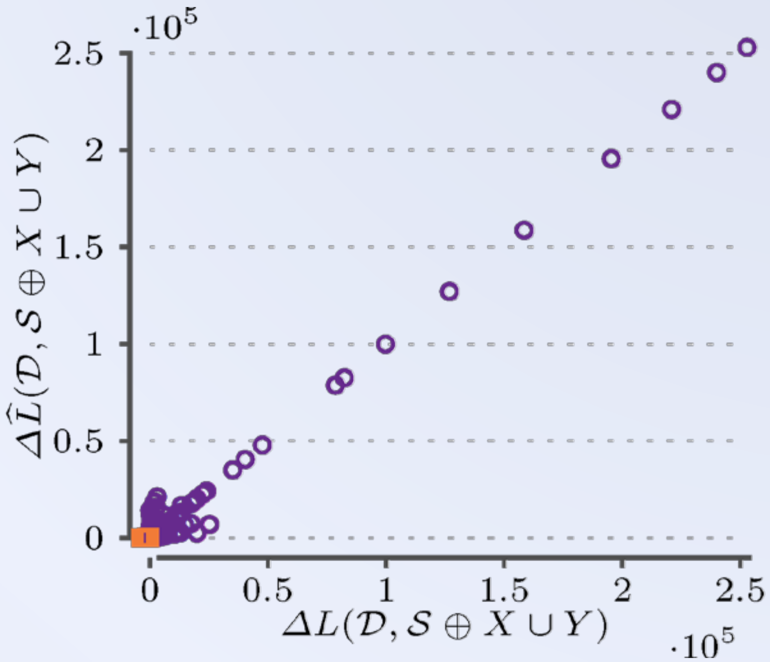
DIFFNORM in action (8)



The Experiments



Effective optimisation



Accurate estimation

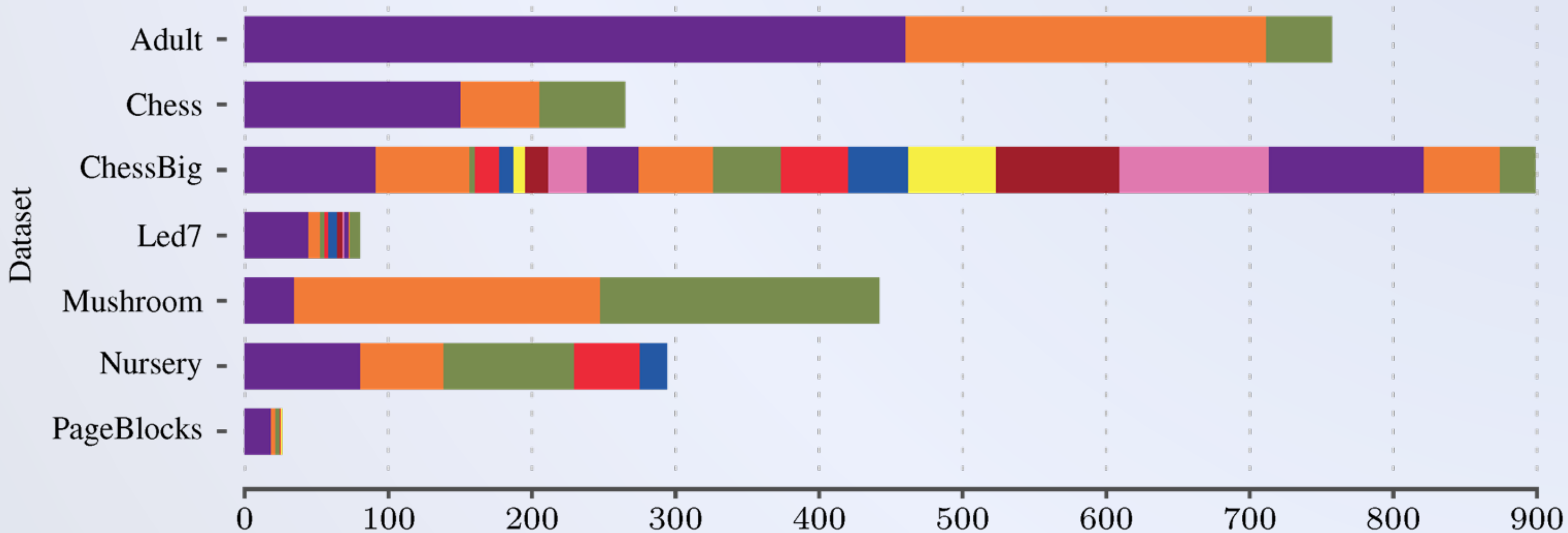
Quantitative Results

First, let's consider some FIMI datasets.
We run DiffNorm to mine an S_i per class, and a global S_Ω

Dataset	$\|\mathcal{D}\|$	$\|\mathcal{J}\|$	$L\%$	$time(s)$	$\|\mathcal{S}\|$
Adult	48842	2	25.6	74	757
ChessBig	28056	18	75.3	11	899
Nursery	12960	5	58.3	7	294
Mushroom	8124	2	25.8	17	442
PageBlocks	5473	5	4.3	0	26
Chess	3196	2	20.6	8	265

Quantitative Results

How are the discovered patterns distributed over $\mathcal{S}$?



Number of patterns per pattern set.
Leftmost (purple) bar indicates size of S_Ω

Qualitative Results

Do the discovered patterns make sense?

We consider abstracts of ICDM as data, splitting on `mining'.

S_{mining}

assoc. rule large datab.
fp tree
prune previous
freq. pat. discover strat.
support threshold

$S_{\neg mining}$

accuracy learn work
svm machine
cluster partition
classifier train
approach learn

S_{Ω}

problem algo. exp. res.
framew. general model
method large set
state [of the] art
evaluation technique

(selection from top-most ranked results)

Meaningless Comparison

How do these numbers compare to when we mine $\mathcal{D}$ globally?

Dataset	$\lvert\mathcal{S}\rvert$		
	DiffNorm($\mathcal{D}$)	DiffNorm($\mathcal{D}_\cup$)	Slim($D_\cup$)
Adult	757	782	2702
ChessBig	899	769	1420
Nursery	294	371	308
Mushroom	442	435	1667
PageBlocks	26	48	105
Chess	265	264	653

(Slim by Smets & Vreeken 2012)

Conclusions

When you have multiple databases,
you want a succinct summary of **difference and norm**

- existing methods are highly restricted, and results redundant
- we formalise the problem in terms of MDL

DIFFNORM

- first attempt for multivariate real-valued data
- non-parametric, somewhat simplistic, yet works very well

Ongoing

- how deep does the rabbit hole go?

Thank you!

Causal inference by **algorithmic complexity**

- solid foundations, clear interpretation, non-parametric
- for *any* pair of **objects** of *any* sort
- for **type** and **token** causation

ERGO

- first attempt for multivariate real-valued data
- non-parametric, somewhat simplistic, yet works very well

Ongoing

- how deep does the rabbit hole go?