

# Topic I.2: Pattern Sets that Compress

Discrete Topics in Data Mining  
Universität des Saarlandes, Saarbrücken  
Winter Semester 2012/13

# **TI.2 Pattern Sets that Compress**

## **1. The MDL Principle Revisited**

### **1.1. The Principle & Motivation**

### **1.2. Kolmogorov Complexity**

## **2. Some Real-World Encodings**

### **2.1. Shannon Entropy**

## **3. The Krimp Algorithm**

### **3.1. Motivation**

### **3.2. Code Tables and Encodings**

### **3.3. The Algorithm**

### **3.4. Some comparisons**

# The MDL Principle – Revisited

- The Minimum Description Length (MDL) principle:  
*The best model is the one that leads to the best compression*
  - A formalization of "simplest" in *Occam's Razor* using bits
- In order to measure how well a model compresses, we need to consider two parts
  - The explanation of the model itself ( $L(\mathcal{M})$ )
  - The explanation of the data given the model ( $L(D \mid \mathcal{M})$ )
  - This is so-called two-part (crude) MDL; in *refined* MDL we encode model and the data together

# The Kolmogorov Complexity

- The length of the shortest program that produces some string  $x$  is the **Kolmogorov complexity** of  $x$ ,  $K(x)$ 
  - Depends on how the program itself is encoded
    - Only constant effect, usually ignored
  - Program can obviously be a general program encoded with some input, but doesn't have to be
- The optimum model (over all possible ones) is then the Kolmogorov program of the data
  - Unfortunately, Kolmogorov complexity of a string (and hence the program to generate it) is *uncomputable*

# The Uncomputability of $K(x)$

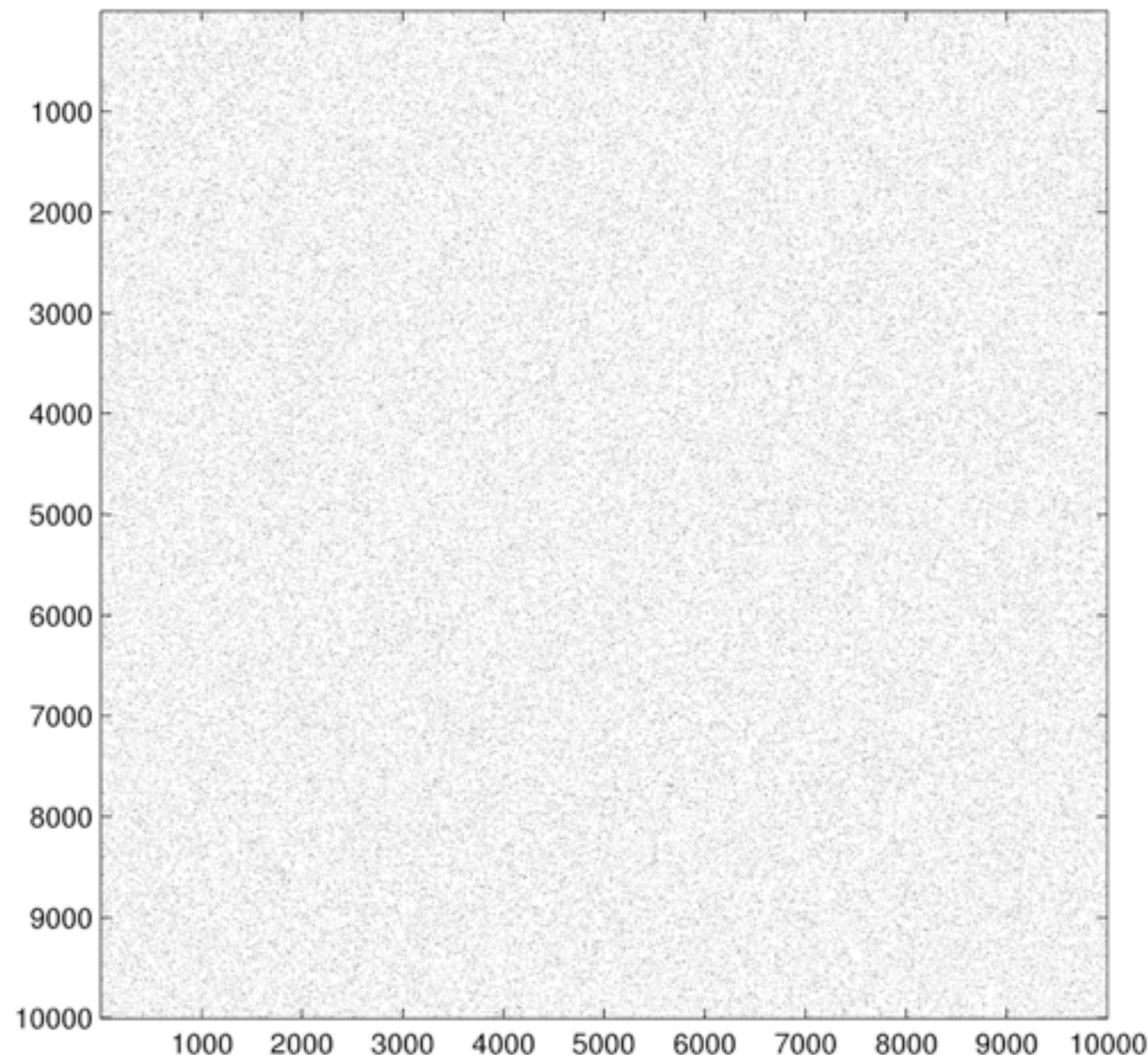
- **Proposition.** For any integer  $n$ , there is always a string  $x$  with  $K(x) > n$ 
  - Otherwise we could express infinite number of strings with finite number of programs
- Let  $P$  be a program that, given  $n$ , returns a string  $x$  with  $K(x) > n$
- Let  $Q$  be a program that calls  $P$  with parameter  $c$  and returns what  $P$  returns
  - Let  $c$  be such that  $c > C + \log_2(c)$ , where  $C$  is the Kolmogorov complexity of  $P$  and  $Q$
- If  $x = Q()$ , then  $K(x) > c$  (output of  $P$ ) and  $K(x) < c$  (can be described with  $P$ ,  $Q$ , and  $c$ )

# The MDL Principle and Data Mining

- The MDL principle can be used to combat *overfitting*
  - Overfitting: model explains the training data too well and doesn't generalize to unseen data
  - MDL presents a natural penalty to too complex models
- The MDL principle can be used to *select* the output
  - Among many possible sets of results (models), select the one that compresses the data best
  - Note: we must explain the *whole* data
    - E.g. MDL does not allow lossy compression
    - But we can circumvent this by having a lossy model and a correction term (error)

# Some Real-World Encodings

- Let's consider the following task: how to encode a binary  $n$ -by- $m$  matrix
- Here's 10000-by-10000 matrix with 5% 1s



# Encoding Integers

- First we have to encode the size of the matrix ( $n, m$ )



# Encoding Integers

- First we have to encode the size of the matrix  $(n, m)$
- First attempt: use 32 bit integers

# Encoding Integers

- First we have to encode the size of the matrix  $(n, m)$
- First attempt: use 32 bit integers
  - But what if  $n > 2^{32}$ ?

# Encoding Integers

- First we have to encode the size of the matrix  $(n, m)$
- First attempt: use 32 bit integers
  - But what if  $n > 2^{32}$ ?
- Second attempt: First encode the length of  $\log_2(n)$  in unary, add 0, and then the binary representation of  $n$

# Encoding Integers

- First we have to encode the size of the matrix  $(n, m)$
- First attempt: use 32 bit integers
  - But what if  $n > 2^{32}$ ?
- Second attempt: First encode the length of  $\log_2(n)$  in unary, add 0, and then the binary representation of  $n$ 
  - This takes  $2\log_2(n)+1$  bits

# Encoding Integers

- First we have to encode the size of the matrix  $(n, m)$
- First attempt: use 32 bit integers
  - But what if  $n > 2^{32}$ ?
- Second attempt: First encode the length of  $\log_2(n)$  in unary, add 0, and then the binary representation of  $n$ 
  - This takes  $2\log_2(n)+1$  bits
- Third attempt: Iteratively re-encode the magnitude using a refined approach

# Encoding Integers

- First we have to encode the size of the matrix  $(n, m)$
- First attempt: use 32 bit integers
  - But what if  $n > 2^{32}$ ?
- Second attempt: First encode the length of  $\log_2(n)$  in unary, add 0, and then the binary representation of  $n$ 
  - This takes  $2\log_2(n)+1$  bits
- Third attempt: Iteratively re-encode the magnitude using a refined approach
  - E.g. Elias Delta coding that takes
$$\lfloor \log_2(n) \rfloor + 2\lfloor \log_2(\lfloor \log_2(n) \rfloor + 1) \rfloor + 1$$
bits

# Encoding the Values: Indices

- If we have a sparse matrix, we can encode the locations of 1s by sending the index pairs
- As we know  $n$  and  $m$ , we can encode the row in  $\log_2(n)$  bits and column in  $\log_2(m)$  bits
  - Or we could again use the Elias Delta, but it can be less effective now that we know the size
- Consequently, we need  $nnz(\mathbf{M})(\log_2(n) + \log_2(m))$  bits

# Interlude: Shannon Entropy

- The **Shannon entropy** of a discrete random variable  $X$  with values  $\{x_1, x_2, \dots, x_n\}$  and probability mass function  $P$  is

$$H(X) = \mathbf{E}[-\log P(X)] = \sum_{i=1}^n P(x_i) \log \frac{1}{P(x_i)}$$

- The binary entropy function for Bernoulli( $p$ ) r.v. is

$$H_b(p) = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

- Assuming a string of i.i.d. random variables, the entropy of the string is a lower bound to the amount of bits needed to represent a symbol in the string exactly (lossless compression) in the limit
  - Entropy is maximal with uniform distributions



# Encoding the Values: Prefix Codes

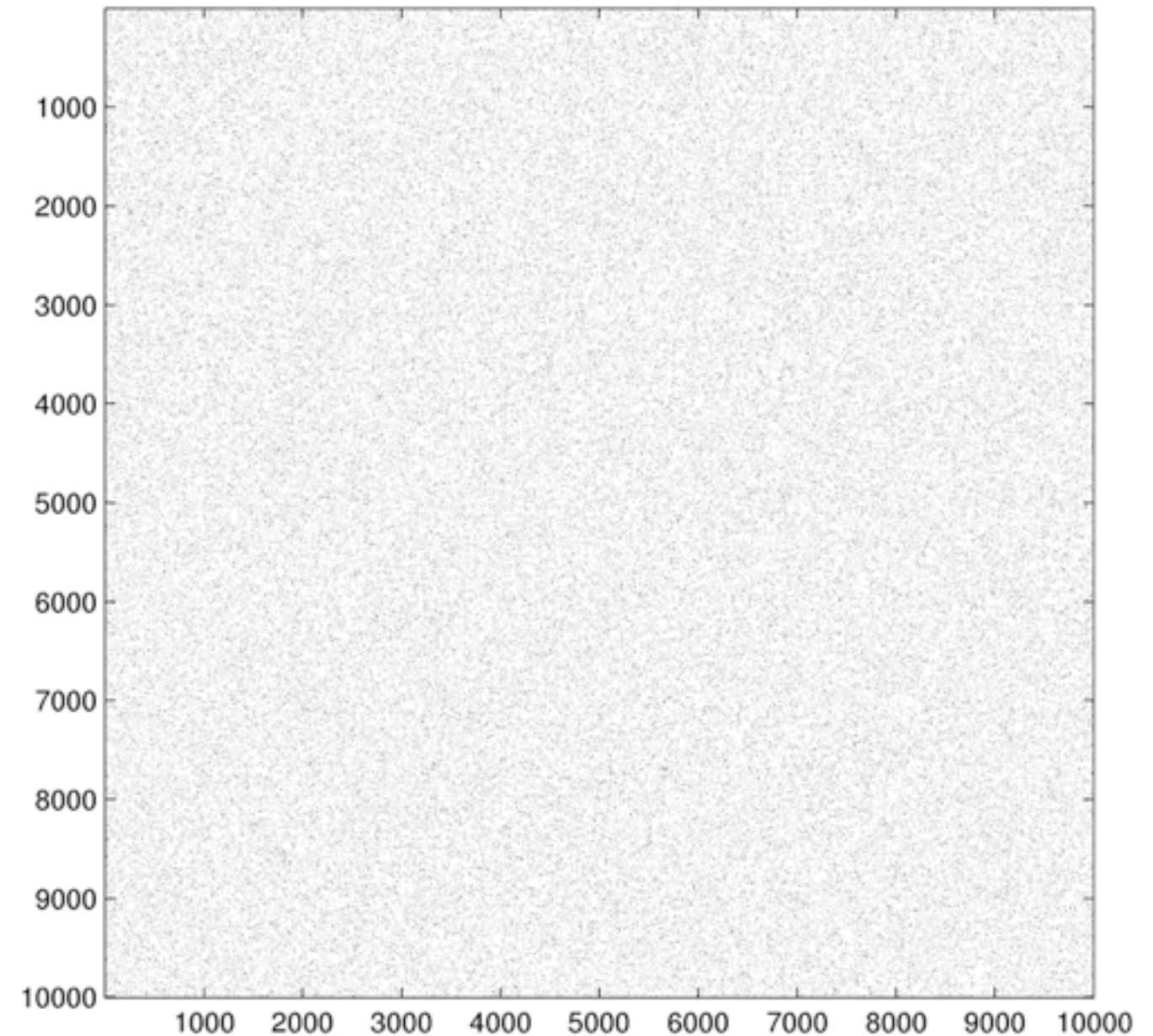
- We can leverage the entropy to give good codes
- Consider the matrix as an  $nm$ -dimensional binary vector
  - If  $p = |D|/mn$  is the fraction of 1s in the data, we should need only  $-|D|\log_2(p) - (mn - |D|)\log_2(1 - p)$  bits
    - Assuming we can use fractional bits...
- To be able to decode the data, we have to use *prefix codes*
  - In prefix codes, every symbol is encoded using a string that has no prefixes that are valid encodings
    - E.g. Huffman coding

# Encoding the Values: Numbering

- We again treat the matrix as a long binary vector
- Count the number of 1s (denote by  $k$ )
- Enumerate all  $nm$ -dimensional binary vectors with  $k$  1s (in lexicographical order, say)
  - There are  $\binom{n}{k}$  such vectors
- Send the number (encoded in binary)
  - This takes  $\log_2 \left( \binom{n}{k} \right)$  bits plus  $\log_2(n)$  for encoding  $k$

# Encodings with the example data

	Encoding length (rounded)
Indices	132932746
Prefix Code	28648613
Numbering	28648600



# The Krimp Algorithm

- The Krimp algorithm utilises the MDL principle to select a set of itemsets to express the data
  - Can be used either with all itemsets of data
  - Or just those above some minimum frequency
- The Krimp covers transactions using itemsets
  - All items in all transactions must be covered
  - No overlapping of coverings is allowed
- The Krimp always returns all singleton items
  - Guarantees the whole data can be explained (covered)
  - The singletons can be interpret as noise









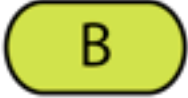




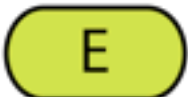

# Encoding the Data: Code Tables

- To encode the data, Krimp uses a *code table*
- A code table is a table with two columns, *key* and *code*
  - To compress, we replace keys with their codes and vice versa to decompress
  - Here, keys are itemsets and codes are the compressed representations of them
- The code table in Krimp always contains every singleton itemset
  - But not all of them have to have a code assigned

# An Example Code Table

$$\mathcal{I} = \{ A, B, C, D, E \}$$

## Code Table

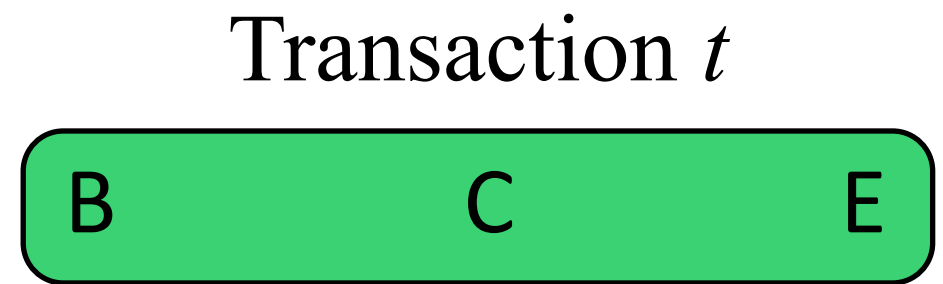
<i>Itemset</i>	<i>Code</i>
	
	
	
	
	
	-
	
	

# Encoding with the Code Table

- The rows in the code table are ordered descending:
  - First on size of the itemset, then on the support, and finally lexicographically
  - Singletons are always at the bottom
- *The keys used must not overlap*
  - For performance reasons
- The keys to cover transaction  $t$  using code table CT are selected as follows:
  - Pick the first key  $X$  in CT for which  $X \subseteq t$  (call this  $Y$ )
  - If  $t \setminus Y$  is empty, return  $Y$ , else return  $Y$  together with the output of a recursive call using  $t \setminus Y$

# Example

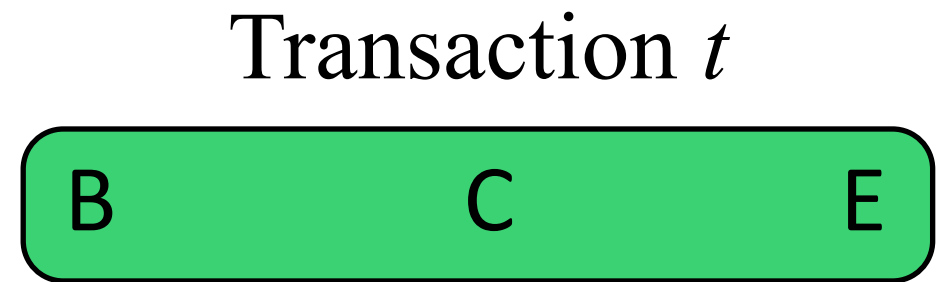
Itemset	Usage
A C	0
B D	0
C E	0
A	0
B	0
C	0
D	0
E	0





# Example

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



# Example

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

Transaction  $t$



Cover of  $t$



# Example

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

Transaction  $t$

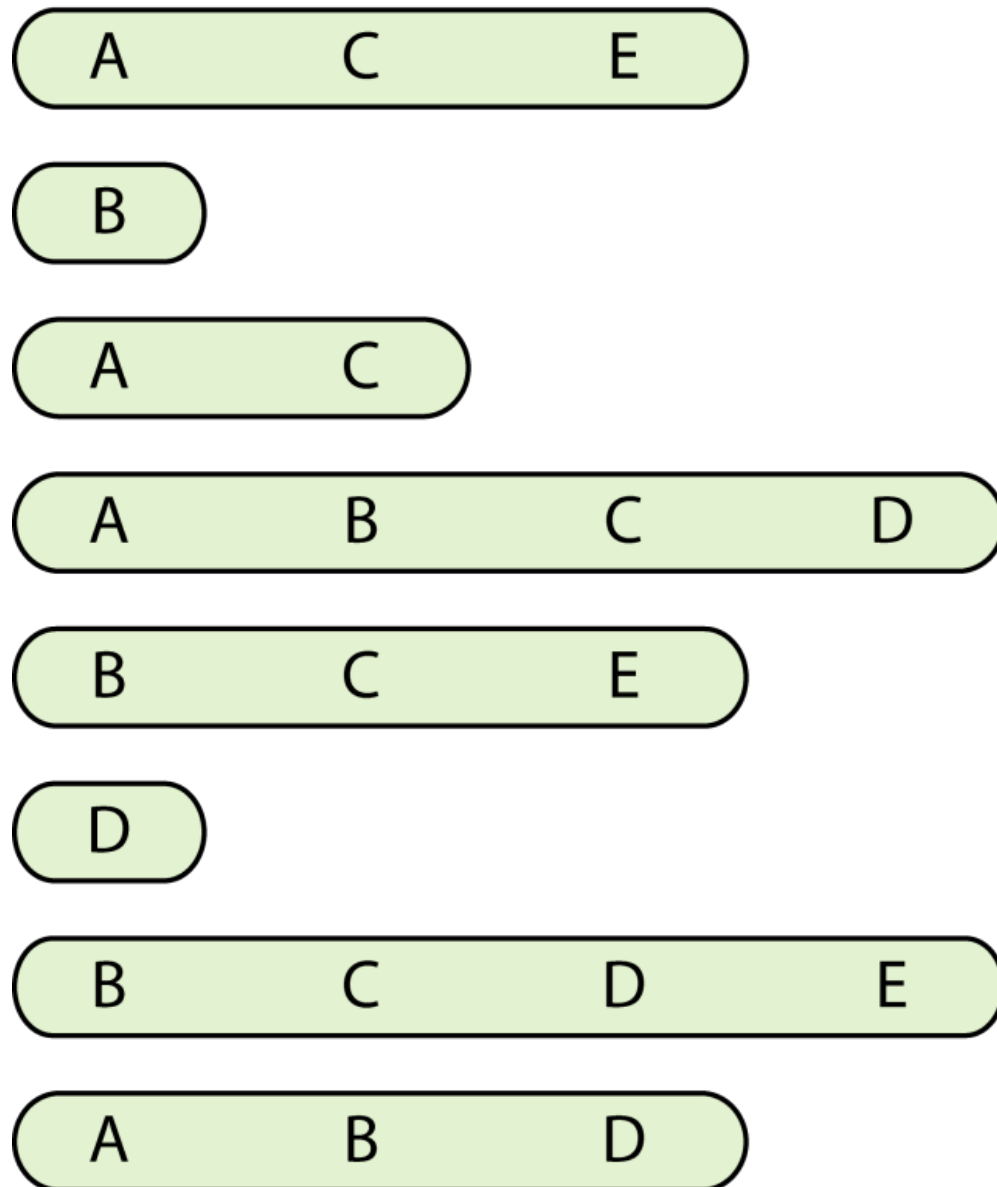


Cover of  $t$

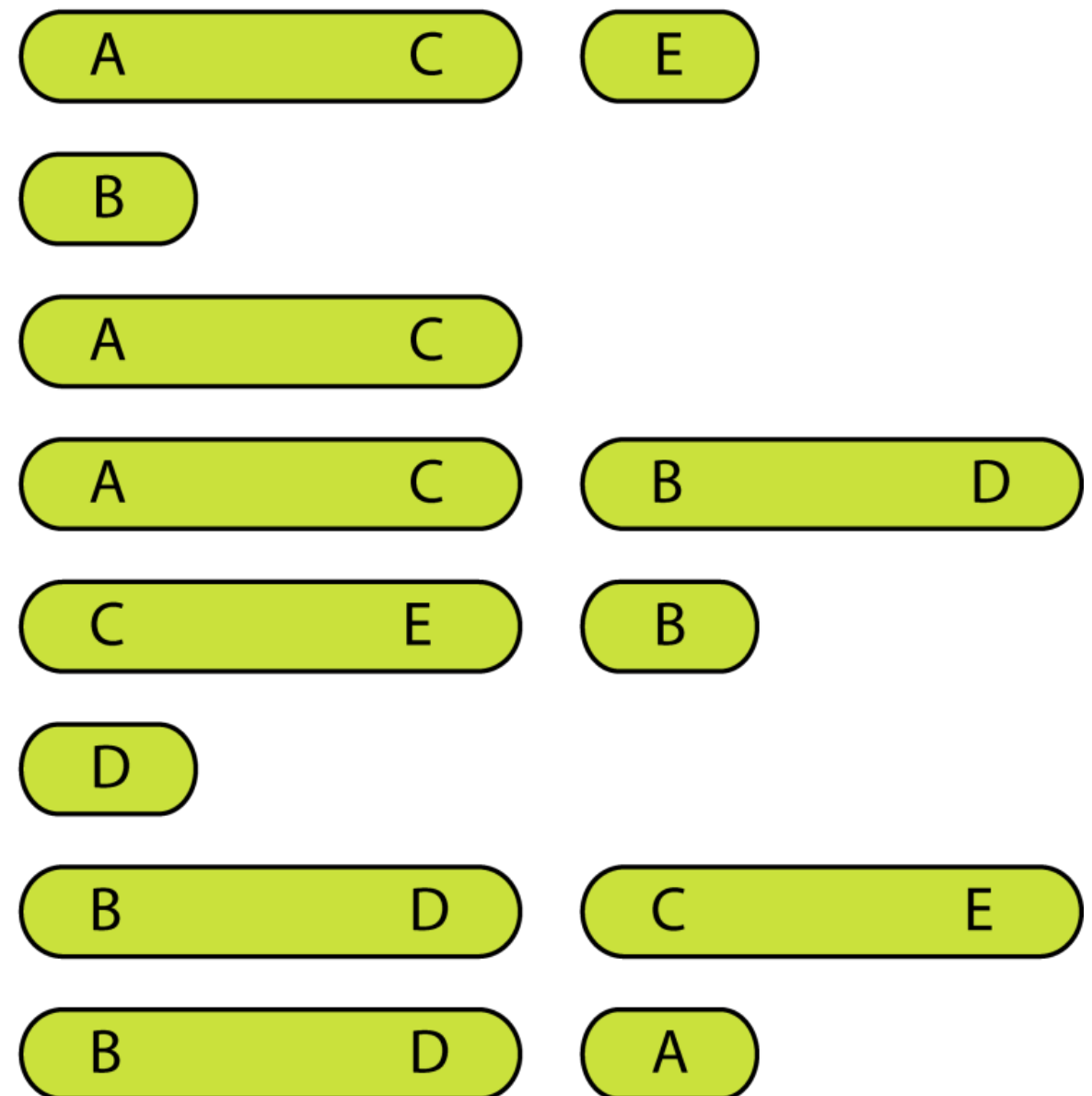


# Example of a Database Cover

## Database










## Database Cover



# Example of a Final Code Table

**Code Table**

<i>Itemset</i>	<i>Code</i>	<i>Usage</i>
A C		3
B D		3
C E		2
A		1
B		2
C	—	0
D		1
E		1

# How to Compute the Codes

- For itemset  $X \in \text{CT}$ , define  $usage(X)$  to be the number of transactions we used  $X$  to cover
  - Define the *probability* of  $X$ ,  $P(X)$ , to be the fraction of  $usage(X)$  of the sum of  $usage(Y)$ 's over all  $Y \in \text{CT}$ ,  
$$P(X) = usage(X) / \sum_{Y \in \text{CT}} usage(Y)$$
- The optimal code to code distribution  $P$  assigns a code  $code(X)$  to  $X \in \text{CT}$  with length  
$$L(code(X)) = -\log(P(X))$$
  - We can use prefix codes for (near) optimal coding
  - But we don't need to realize any coding, we only need the length

# The Length of a Code Table

- The code table is our model, which we need to encode
  - The prefix codes are ready as-is
- To encode the itemsets, we first encode the singletons using just their frequencies
  - Called *standard code table*, ST
  - We can just give the codes in order
    - Or we can ignore them, as they are in every code table
- Then we encode the itemsets with non-zero usage using ST
- Therefore: 
$$L(\text{CT}) = \sum_{\substack{X \in \text{CT} \\ \text{usage}(X) \neq 0}} (L_{\text{ST}}(X) + L_{\text{CT}}(X))$$

# The Length of the Data and Everything

- The length of a transaction is

$$L(t \mid \text{CT}) = \sum_{X \in \text{cover}(t)} L(\text{code}(X))$$

- The length of the whole data is

$$L(D \mid \text{CT}) = \sum_{t \in D} L(t \mid \text{CT})$$

- The overall description length then is

$$L(D, \text{CT}) = L(\text{CT}) + L(D \mid \text{CT})$$



# The Krimp Algorithm

- **Problem.** Given a data, find the code table that minimizes the description length.
- Quite a large search space
  - Because we have to have the singletons, the number of possible keys in code table is
  - For one transaction over  $I$ , the number of possible ways to cover it is

$$\sum_{k=0}^{2^{|I|}-|I|-1} \binom{2^{|I|}-|I|-1}{k} = 2^{2^{|I|}-|I|-1}$$

$$\sum_{k=0}^{2^{|I|}-|I|-1} \binom{2^{|I|}-|I|-1}{k} \times (k + |I|)!$$

# Krimp: A Sketch

1. Cover everything with singletons
2. Consider itemsets one-by-one
  - 2.1. Add the itemset to the code table
  - 2.2. If encoding length reduces, keep it, else discard it
- Questions
  - In which order do we consider the itemsets?
  - How the itemsets are used to cover the data?
  - Should we prune the code table after adding a new itemset?

# Some Details

- The order to consider itemsets
  - Descending on support, then size, then lexicographically
    - High-support itemsets are intuitively the ones with short codes
- How to cover
  - Keep the itemsets in code table sorted by length (then support)
  - After adding a new itemset, re-compute the cover for the data (and the coding lengths)
- Finally, we can do some pruning after adding a new itemset to the code table
  - Not straight forward; see the paper

# Time Complexity

- If  $f$  is the number of frequent itemsets,  $d$  is the number of transactions and  $i$  is the number of items, Krimp without pruning takes

$$O(f \log f + f \times (d f i + f))$$

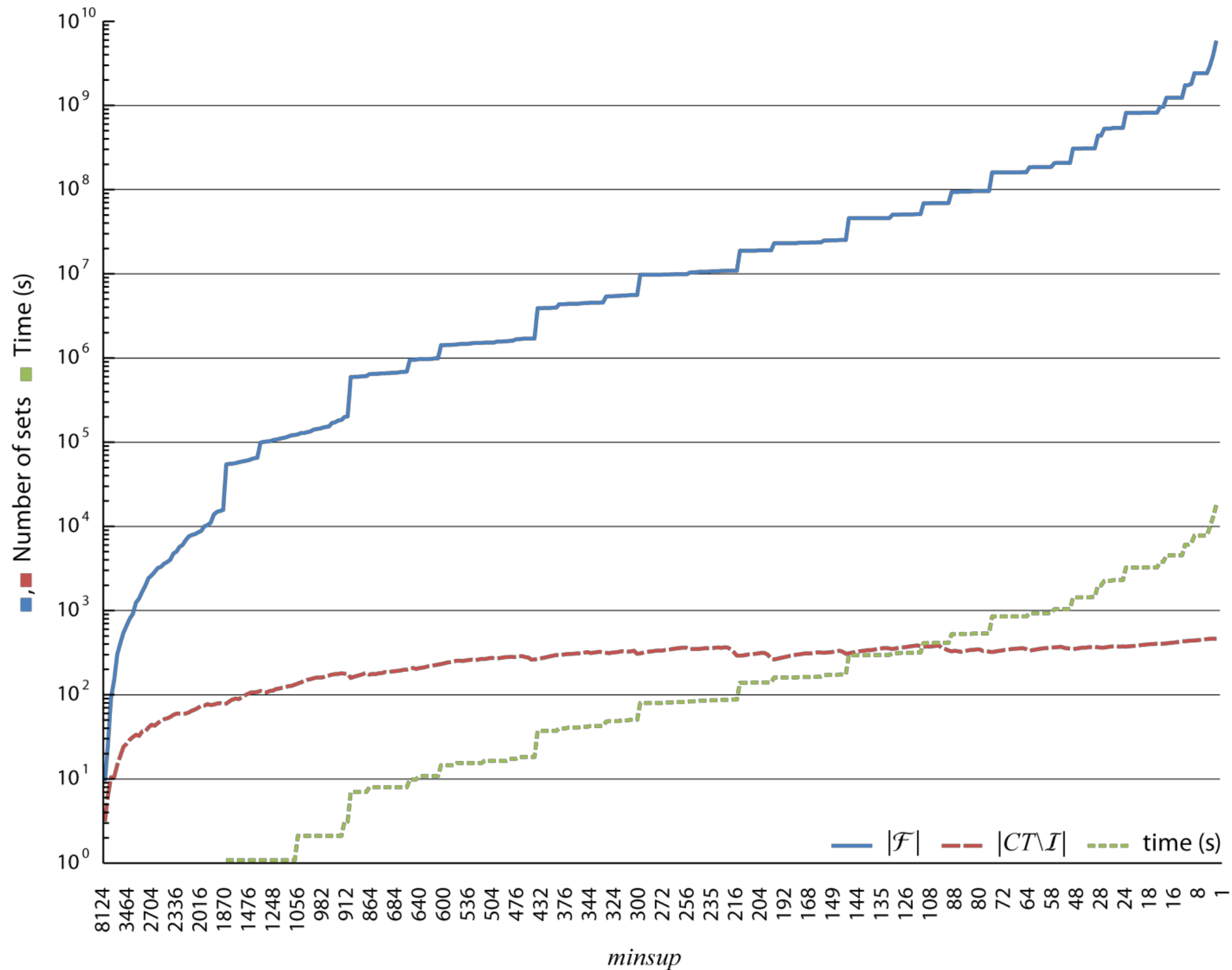
- Quadratic in the number of frequent itemsets!

- But that estimates the size of the code table by  $f$

- If code table is of constant size, we get

$$O(f \log f + d f i)$$

# Some Results



# More Results

Dataset	$ \mathcal{D} $	$ \mathcal{F} $	$ \mathcal{CT} \setminus \mathcal{I} $	$L\%$
Accidents	340183	2881487	467	55.1
Adult	48842	58461763	1303	24.4
Letter Recog.	20000	580968767	1780	35.7
Mushroom	8124	5574930437	442	24.4
Wine	178	2276446	63	77.4

# Essay Topics

- Choose one of the following
  - The hints below give you idea how to approach the question; you should cover other questions than just those (and you don't necessarily cover all of them)
  - Justify your opinions and give arguments!
- 0/1 Tiling versus Density Tiling
  - Pros and cons of both methods? When they can be used? When they should be used? When one is better than the other? Can we use one to get other? Better algorithms?
- 0/1 Tiling versus Krimp
  - How the approaches differ? Pros and cons? Can we use parts of one in other (e.g. MDL in tiling? Set Cover in Krimp?)?
- MDL versus Bayesian Information Criterion
  - Requires extra reading
  - Differences/similarities? Pros and cons? When one is better than the other? Which one should I use?

# Some feedback from the 1st Essays

- Overall good essays, but...
  - Remember to give good justification for your arguments
    - Tell what others said and why and then tell what you think and why
    - Are these others reliable? Why? Why not?
  - Remember to cite! (within the text)
    - On-line sources require: Author (if known), title, URL, and access date
    - Google queries are not valid sources!
  - Think big and think small!
    - All of you seemed to think data mining as a bag of tricks...