Mining patterns in collections of structured data

CM4 – Optimization based pattern mining

Thomas Guyet

Inria – AIstroSight

M1 ENS – DBDM – 2023

This presentation uses slides/works borrowed from

- Jilles Vreeken and Peggy Cellier

Outline

Pattern mining and optimization

Pattern mining and constraint programming

→ Use of CP solvers achieve the task

→ Representation of the pattern mining problem with constraint programming tools

Pattern mining and information theory

→ Select subset of patterns through information theory (MDL principle)

→ Representation of the pattern mining as a coding of data

Today ... we play lego!

Outline

Frequent pattern extraction: algorithm principles

Two important problems for pattern mining

1. search space traversal
   - ⇒ use anti-monotonicity property to prune the search space
   - ⇒ search strategy efficient and without redundancy
2. evaluate the support
   - ⇒ enumeration method (simple for itemsets, but may be complex, *e.g.* sequences or graphs)
   - ⇒ reduce the database scans
   - ⇒ split database in sub-databases (for parallelism)

Toward a generic approach of pattern mining I

Toward tools for *better* pattern extraction

- needs for user focus on most interesting patterns

⇒ use formalized knowledge

- needs for more versatility

⇒ develop generic tools for pattern mining

Toward generic tools

- mining structured patterns with different shapes (itemsets, sequences, trees, graphs)
- easy insertion of formalized knowledge into the mining process
  - benefit from expert knowledge during the mining process
  - a posteriori (*post-mining*)
- integrate tools into a all-in-one generic data mining platform

Toward a generic approach of pattern mining II

Various approaches

- Algorithmic approach: reduce mining problem to well-known problem classes
  - Problem class example: Frequent itemset class (very well-known, very efficient algorithms), large number of pattern mining task can be reduced to this case
  - How to generalize this approach to other classical mining tasks (?)
- Constraint based approaches ⇒ SAT, CP, ASP

Using ASP (Answer Set Programming) for pattern mining?

- language expressiveness: easy for integrating knowledge and reasoning (the nexy step)
- efficiency of state-of-the art solvers (`clingo`)

Answer Set Programming I

ASP programs

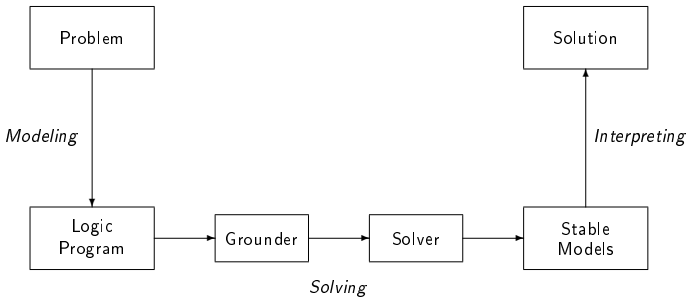
- ideal of declarative programming: *the problem is the program*
  - ⇒ modeling of the problem to be solved in the form of rules and constraints expressed in a logical language;
- the solution is a set of model assignments (so-called answer-set)
  - ⇒ a model = an answer set
  - ⇒ formalized by Herbrand models (boolean assignment to variables)
- constraints on sets of atoms can be expressed (and efficiently solved)

⇒ a (*magic*) solver (efficiently) finds one, some or all answer sets

`clingo` solver

- A reference solver (among others like: DLV or Cmodels)
- `https://potassco.org/`: many resources to learn ASP

Answer Set Programming II



Two-step solving schema specific to `clingo`

ASP Programming

Programming principles

1. set of rules: "generate" boolean assignments of atoms (potential answer sets)
2. set of constraints: prune undesirable assignments
3. printing directives: `#show`

Graph coloring

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2). edge(1,3). edge(1,4). edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5). edge(4,1). edge(4,2). edge(5,3).
edge(5,4). edge(5,6). edge(6,2). edge(6,3). edge(6,5).
col(r). col(b). col(g).

1 { color(X,C) : col(C) } 1 :- node(X).
:- edge(X,Y), color(X,C), color(Y,C).

#show color/2.
```


Resolution principle – *grounding* I

- Program P

$$p(X,Y) \leftarrow q(X,Y). \quad q(a,b). \quad q(b,c).$$

- Herbrand's universe $H = \{a, b, c\}$
- Theoretical grounded program (replace variables by constants)

$$\left\{ \begin{array}{lll} p(a,a) \leftarrow q(a,a), & p(a,b) \leftarrow q(a,b), & p(a,c) \leftarrow q(a,c), \\ p(b,a) \leftarrow q(b,a), & p(b,b) \leftarrow q(b,b), & p(b,c) \leftarrow q(b,c), \\ p(c,a) \leftarrow q(c,a), & p(c,b) \leftarrow q(c,b), & p(c,c) \leftarrow q(c,c), \\ q(a,b) & q(b,c) & \end{array} \right\}$$

- Grounding by `gringo`

$$\{p(a,b), p(b,c), q(a,b), q(b,c)\}$$

Resolution principle – *grounding* II

- All variables in a rules have to be "grounded", otherwise the variable is *unsafe*.

```
:- node(X), not color(X,C). % C is unsafe
1 { color(X,C) } 1 :- node(X). % C is unsafe !
```

- To ensure that, all rules' variables have to have a finite domaine

```
:- node(X), col(C), not color(X,C).
1 { color(X,C) : col(C) } 1 :- node(X).
```

- `gringo` attempt to infer the smaller domain for each variable to reduce the size of the grounded program

⇒ a grounded program can be seen as a classical boolean constraints program

Warning

Bad rules may lead to huge grounding ... (primer: this is the main problem of that approach!)

Resolution principle – *solve*

- Once grounded: all variables are boolean ⇒ "classical constraints" programming (CP)!

→ The problem to solve is to find a complete assignment of the boolean variables that satisfies all the constraints.

Boolean solver

- A classical approach is back-tracking
  1. add variable assignments to a current partial assignment
  2. evaluate the constraints
  3. if satisfied, then continue, otherwise backtrack
- The `clingo` uses a solver called *clasp*
- *clasp* is a CDCL solver (efficient strategy for CP solvers)

Resolution principle – *solve*

Conflict Driven Resolution – *clasp*

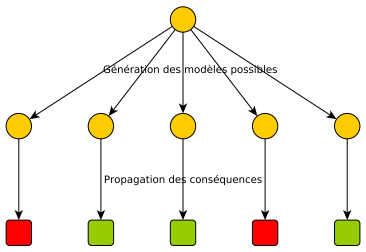
Assigned=T
WHILE Assigned

- Propagate deterministic consequences
- IF no conflicts (inconsistency between consequences and assignments)
  - IF all variables are assigned
    - Output the solution
    - Assigned=F
  - ELSE
    - Choose one variable and assign it a truth-value
- ELSE
  - Analyse the conflict: extract the minimal set of assigned variables to define the conflict (*nogood*)
  - Add a constraint
  - Unset the assigned variable

Resolution principle – a practical view

Warning

This slide simply gives the intuition of the solving process. See solver details for understanding the know how.

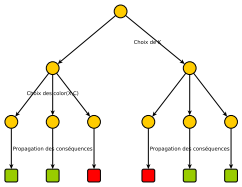


Resolution principle – a practical view

- The combinatorial explosion of the search space is managed by the solver!
- But, be carefule with problem symmetries (redundant solutions):
  - ▷ **there is no shared-information from one model to another!**
  - ▷ image of distributed solving

Find the minimal number of colors to color a graph.

```
1{ nb_colors(1..6) } 1.
K{ col(a;m;b;c;g;o) } K :- nb_colors(K).

1 { color(X,C) : col(C) } 1 :- node(X).
:- edge(X,Y), color(X,C), color(Y,C).

#minimize{ K : nb_colors(K) }.

#show color/2.
```



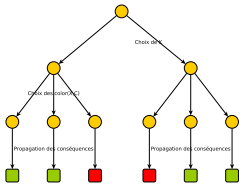
Intro ASP pattern mining Information theory CCL on pattern mining

Resolution principle – a practical view

- The combinatorial explosion of the search space is managed by the solver!
- But, be carefule with problem symmetries (redundant solutions):
  - ▷ **there is no shared-information from one model to another!**
  - ▷ image of distributed solving

Find the minimal number of colors to color a graph.

```
1{ nb_colors(1..6) } 1.
K{ col(a;m;b;c;g;o) } K :- nb_colors(K).

1 { color(X,C) : col(C) } 1 :- node(X).
:- edge(X,Y), color(X,C), color(Y,C).

#minimize{ K : nb_colors(K) }.

#show color/2.
```



⇒ This encoding is bad because of the redundancy of the solutions (same solutions with different colors' names)

ASP Programming and Pattern Mining

Programming principles – related to pattern mining tasks

1. set of rules: "generate" boolean assignments of atoms (potential answer sets) ← **version space traversal!**
2. set of constraints: prune undesirable assignments ← **embedding and support evaluation**
3. printing directives: `#show`

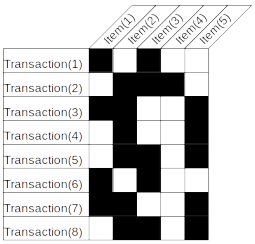
Graph coloring

```
node(1). node(2). node(3). node(4). node(5). node(6).
edge(1,2). edge(1,3). edge(1,4). edge(2,4). edge(2,5). edge(2,6).
edge(3,1). edge(3,4). edge(3,5). edge(4,1). edge(4,2). edge(5,3).
edge(5,4). edge(5,6). edge(6,2). edge(6,3). edge(6,5).
col(r). col(b). col(g).

1 { color(X,C) : col(C) } 1 :- node(X).
:- edge(X,Y), color(X,C), color(Y,C).

#show color/2.
```


Frequent itemset mining with ASP

First attempt of encoding the itemset mining

- 1 model = 1 pattern
- useful predicates:
  - **db**/2 : database
  - **support**/1 : the pattern supports a transaction
  - **item**/1 : items (*optional*)
  - **transaction**/1 : transactions (*optional*)

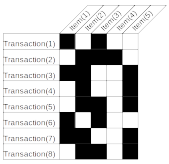


Frequent itemset mining with ASP

Encoding proposed by Järvisalo [Jär11]

```
item(I) :- db(_,I). % set of items in the database
transaction(T) :- db(T,_). % set of transaction in the database

{ in_itemset(I) } :- item(I). % generation of all possible itemset patterns
in_support(T) :- { conflict_at(T,I) : item(I) } 0, transaction(T).
conflict_at(T,I) :- not db(T,I), in_itemset(I), transaction(T).
:- { in_support(T) } N-1, threshold(N).
```

- item/1 : items
- transaction/1 : transactions
- db/2 : database



Frequent itemset mining with ASP

Encoding proposed by Järvisalo [Jär11]

```
item(I) :- db(_,I). % set of items in the database
transaction(T) :- db(T,_). % set of transaction in the database

{ in_itemset(I) } :- item(I). % generation of all possible itemset patterns
in_support(T) :- { conflict_at(T,I) : item(I) } 0, transaction(T).
conflict_at(T,I) :- not db(T,I), in_itemset(I), transaction(T).
:- { in_support(T) } N-1, threshold(N).
```

- 1 AS = 1 motif
- modeling inspired by [GNR11] : conflict_at/2 encodes that a pattern is not supported by a transaction T

Frequent itemset mining with ASP

Encoding proposed by Järvisalo [Jär11]

```
item(I) :- db(_,I). % set of items in the database
transaction(T) :- db(T,_). % set of transaction in the database

{ in_itemset(I) } :- item(I). % generation of all possible itemset patterns
in_support(T) :- { conflict_at(T,I) : item(I) } 0, transaction(T).
conflict_at(T,I) :- not db(T,I), in_itemset(I), transaction(T).
:- { in_support(T) } N-1, threshold(N).
```

What's about the encoding of anti-monotony ?

- No ... it is not a good idea to encode this "expert constraint"
- The solver "learns" the anti-monotony (through conflicts learning)

Other classical pattern mining constraints

- maximal and closed
- experiments on the classical itemset databases

```
item(I) :- db(_,I).
transaction(T) :- db(T,_).

{ in_itemset(I) } :- item(I).
in_support(T) :- { conflict_at(T,I) : in_itemset(I) } 0, transaction(T).
conflict_at(T,I) :- not db(T,I), in_itemset(I), transaction(T).
:- { in_support(T) } N-1, threshold(N).

%closed patterns rule
:- {not db(T,I) : in_support(T)} 0, item(I), not in_itemset(I).

%maximal patterns rule
:- N {in_support(T) : db(T,I)}, threshold(N), item(I), not in_itemset(I).
```

Some results I

Anneal datasets, 812 transactions, 38 items

threshold	time(s)	models
2	10.825	1804942
4	10.707	1802890
20	10.330	1738439
40	9.585	1592867
60	8.797	1414724
100	7.095	1061923
200	3.763	439065

Table: Closed

threshold	time(s)	models
2	1.768	837
4	4.782	2087
20	10.245	8497
40	18.923	12983
60	10.789	15284
100	11.539	15889
200	7.285	10529

Table: Maximals

Some results II

Mushroom dataset, 8124 transactions, 119 items

threshold	time(s)	models
1%	26.881	38120
5%	18.789	8986
10%	16.636	3287
15%	15.774	1535

Table: Closed

threshold	time(s)	models
1%	278.8	5650
5%	52.5	1214
10%	22.3	453
15%	18.6	240

Table: Maximals

⇒ several orders of magnitude less time effificent ... but faster to program

⇒ the main problem is memory: some dense datasets can not be tester

⇒ same performances as the constraint programming approaches CP4IM [GNR11]

→ Itemset mining is a boolean problem: easy to encode! What's about sequence mining!

Declarative sequential patterns

Sequential pattern mining

Sequential pattern mining is a challenge:

- more complex than itemsets mining (can not be reduced to a boolean matrix)
- modeling challenge of complex dataset (time modeling, event modeling)

→ very appropriate to illustrate the declarative pattern mining approach

Sequence database encoding

- seq(T,D,I) : transaction T, at date D, the item I

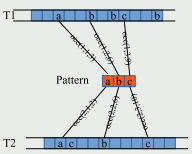
```
seq(0,1,20). seq(0,2,19). seq(0,3,2). seq(0,4,18).
seq(1,1,5). seq(1,2,6). seq(1,3,6). seq(1,4,14).
seq(2,1,13). seq(2,2,12). seq(2,3,9). seq(2,4,19).
seq(3,1,15). seq(3,2,7). seq(3,3,17). seq(3,4,17).
seq(4,1,11). seq(4,2,2). seq(4,3,2). seq(4,4,13).
seq(5,1,14). seq(5,2,8). seq(5,3,13). seq(5,4,1).
```


Sequential pattern I

Encoding inspired by the Järvisalo approach [Jär11].

- patlen/1 pattern length (fixed)
- pat/2 describe the pattern (1 pattern/AS)
- occ/3 occurrences of the pattern
- support/1 transactions that support the pattern

Example: description of transaction occurrences




```
pat(a). pat(b). pat(c).
occ(1,1,3). occ(1,2,6). occ(1,3,9).
occ(2,1,2). occ(2,2,6). occ(2,3,10).
```

Sequential pattern II

```
item(I) :- seq(_, _,I).

%sequential pattern generation
patpos(1).
0 { patpos(Ip+1) } 1 :- patpos(Ip), Ip<maxlen.
patlen(L) :- patpos(L), not patpos(L+1).

1 { pat(Ip,I): item(I) } 1 :- patpos(Ip).

%pattern embeddings
occ(T,1    ,Is):- seq(T,Is,I), pat(1,    I).
occ(T,Ip+1,Is):- seq(T,Is,I), pat(Ip+1,I), occ(T,Ip,Js), Js<Is.

%frequency constraint
support(T) :- occ(T,L,_), patlen(L).
:- { support(T) } < th.
```


Sequential pattern III

Add *maxgap* constraints

(minima/maximal) duration constraint between two item occurrences

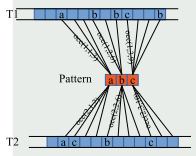
```
%gap constraint
:- occ(I,N,P), pat(N,C), occ(I,N+1,P2),pat(N+1,C2), P2-P>maxgap+1, P2-P<mingap.
```

Code analysis

- complete and correct program
- Occurrence enumeration cost is important (*grounding* phase)
  - ⇒ not surprising, supplementary complexity compared to itemsets
- gap constraint is also costly: increase the grounding cost (dramatically)
- important redundancy in the enumeration of occurrence lists (frequent in different manner)

Alternative encoding: fill-gaps strategy

Description of the occurrences: second solution, the **fill-gaps** solution




```
%pattern embeddings
occF(T,1,Is)   :- seq(T,Is,I), pat(1,I).
occF(T,Ip,Is) :- occF(T, Ip-1, Is-1), seq(T,Is,I),
      pat(L,I).
occF(T,Ip,Is) :- occF(T, Ip,    Is-1), seq(T,Is,_).

%frequency constraint
seqlen(T,L) :- seq(T,L,_), not seq(T,L+1,_).
support(T) :- occF(T, L, LS), patlen(L), seqlen(T,LS).
:- { support(T) } < th.
```

```
pat(a). pat(b). pat(c).
occ(1,1,3). occ(1,1,4). occ(1,1,5). occ(1,2,6). occ(1,2,7)
      . occ(1,2,8). occ(1,3,9).
occ(2,1,2). occ(2,1,3). occ(2,1,4). occ(2,1,5). occ(2,2,6)
      . occ(2,2,7). occ(2,2,8). occ(2,2,9). occ
      (2,3,10).
```


Alternative encodings

Several alternative encodings were implemented[?]

- Alternative strategies
  - Skip-gaps vs fill-gaps embedding evaluations
  - Early filtering out infrequent items
  - Using projected databases principle
- Alternative tasks
  - Constrained sequential patterns
  - Closed and maximal sequential patterns: pure ASP or based on preferences [GGQ+16]
  - Rare sequential pattern mining [?]

Experiments

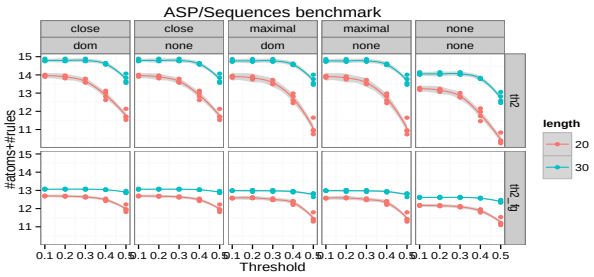
Evaluated encodings

- Two different encodings for occurrences search
  - `sg`: basic encoding
  - `fg`: evaluating occurrences with the *fillgap* strategy
- Constraints: `None`, `Maximal` or `Closed`
- Solving strategy (solver option): `normal`, subset minimal (`dom`) possibly incomplete
- CPSM: Constraint Programming for Sequence Mining [NG15]

Evaluation criteria

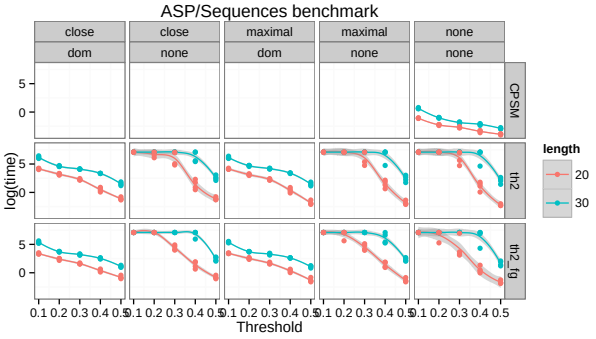
- computing efficiency: time to extract all the patterns
- memory usage: number of atoms grounded

Results – Memory usage



- fill-gaps encoding reduce significantly the required memory
- length of the sequences is a very sensitive dataset characteristic
- very low threshold will not increase the memory requirements
- maximal and closed encoding require more memory

Results - Computing time



- fill-gaps is slightly more efficient (especially for large memory requirement problems)

Conclusions

- pure ASP enables to encode pattern mining tasks in a "intuitive" way
  - ⇒ very attractive for fast prototyping of a task
- several very different encodings can be proposed
  - ⇒ the more efficient ones are much more complex (and not always really intuitive)
- ASP is not the competitive with state of the art algorithms but comparable with "fairly-versatile" other declarative frameworks (SAT/CP)
  - → memory requirement is the harder constraint
  - → some perspectives to improve efficiency using *propagators*

Why is it interesting?

⇒ **versatility**: to be combined with knowledge reasoning

⇒ fast prototyping

Other works related to pattern mining and declarative languages

Declarative pattern mining

- MiningZinc: CP for pattern mining (Guns et al.)
- SAT based pattern mining (Sais L. et al.)
- Pattern set discovery: preference based pattern mining (Cremillieux et al.)

Some challenges

- propose global constraints specific to pattern mining
- deep integration of reasoning and mining

Outline

Standard pattern mining

Reminder: standard old-fashioned pattern mining

For a database $\mathcal{D}$

- a pattern language $\mathcal{L}$
- and a set of constraints $\mathcal{C}$

the goal is to find the set of patterns $P \subseteq \mathcal{L}$ such that

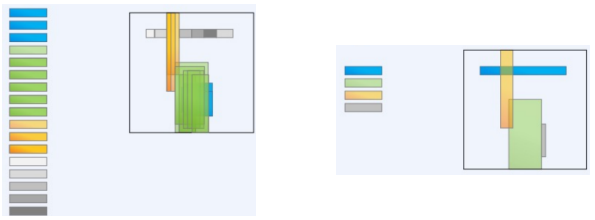
- each $p \in P$ satisfies each $c \in \mathcal{C}$ on $\mathcal{D}$

That is, **find all patterns** that satisfy the constraints

Challenge to make pattern mining practical ...

Less but meaningful patterns !

Intuition to reduce the number of patterns



- a pattern represents a set of transactions
- an interesting set of patterns is such that:
  - the set of pattern is small
  - it *covers* most of the data

⇒ looks like a tiling problem ...

- find a small collection of patterns to tile the whole dataset

⇒ or like a **compression problem**

- we are looking for a dataset reconstruction with few patterns

Loosless compression: Example

Example of lossless compression

Let M be a message as a sequence of integers in $[0, 15]$, for instance

$$M = \langle 10, 10, 10, 15, 13, 10, 12, 12, 12 \rangle$$

. **How to encode this sequence in a binary representation?**

Naive binary representation

Its is naively:

$$1010 - 1010 - 1010 - 1111 - 1101 - 1010 - 1100 - 1100 - 1100$$

with the following binary code table:
Code length:$9 \times 4 = 36$ bits

10	$\leftrightarrow$	1010
12	$\leftrightarrow$	1100
13	$\leftrightarrow$	1101
15	$\leftrightarrow$	1111

Loosless compression: Example

Example of lossless compression

Let M be a message as a sequence of integers in $[0, 15]$, for instance

$$M = \langle 10, 10, 10, 15, 13, 10, 12, 12, 12 \rangle$$

. **How to encode this sequence in a binary representation?**

Coding only 4 values? We can be smarter!

Its is naively:

$$0 - 0 - 0 - 11 - 01 - 0 - 1 - 1 - 1$$

with the following binary code table:
code length: 11 bits !!
but ... this coding is a bad idea!
why?

10	$\leftrightarrow$	0
12	$\leftrightarrow$	1
13	$\leftrightarrow$	01
15	$\leftrightarrow$	11

Loosless compression: Example

Example of lossless compression

Let M be a message as a sequence of integers in $[0, 15]$, for instance

$$M = \langle 10, 10, 10, 15, 13, 10, 12, 12, 12 \rangle$$

. **How to encode this sequence in a binary representation?**

Coding only 4 values with Huffman coding

Its is naively:

$$0 - 0 - 0 - 111 - 110 - 0 - 10 - 10 - 10$$

with the following binary code table:
code length: 16 bits !!

10	$\leftrightarrow$	0
12	$\leftrightarrow$	10
13	$\leftrightarrow$	110
15	$\leftrightarrow$	111

What do you really need?

Example of lossless compression

Let M be a message as a sequence of integers in $[0, 15]$, for instance

$$M = \langle 10, 10, 10, 15, 13, 10, 12, 12, 12 \rangle$$

. **How to encode this sequence in a binary representation?**

Huffing code of the sequence

$$0 - 0 - 0 - 11 - 01 - 0 - 1 - 1 - 1$$

What do you really need to reconstruct the sequence from the code?

What do you really need?

Example of lossless compression

Let M be a message as a sequence of integers in $[0, 15]$, for instance

$$M = \langle 10, 10, 10, 15, 13, 10, 12, 12, 12 \rangle$$

. **How to encode this sequence in a binary representation?**

Huffing code of the sequence

$$0 - 0 - 0 - 11 - 01 - 0 - 1 - 1 - 1$$

What do you really need to reconstruct the sequence from the code?

- the code sequence itself
- but also the code table ...

→ the real encoding length sum the two encodings

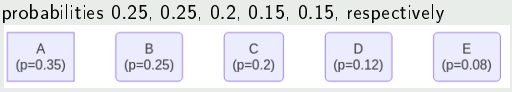
Huffman coding is the optimal coding

Finding optimal prefix-free code

Given probabilities $p_1, p_2, \ldots, p_k$ solve for code-lengths $l_i \in \mathbb{N}$, such that minimize $L(p) = \sum_{i=1}^{k} p_i l_i$

Huffman code construction

Consider a random variable X taking values in the set $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities 0.25, 0.25, 0.2, 0.15, 0.15, respectively



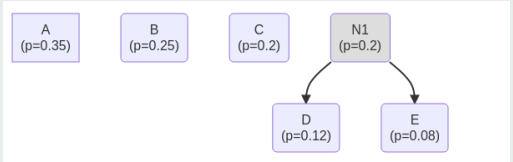
Huffman coding is the optimal coding

Finding optimal prefix-free code

Given probabilities $p_1, p_2, \ldots, p_k$ solve for code-lengths $l_i \in \mathbb{N}$, such that minimize $L(p) = \sum_{i=1}^{k} p_i l_i$

Huffman code construction

Consider a random variable X taking values in the set $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities 0.25, 0.25, 0.2, 0.15, 0.15, respectively



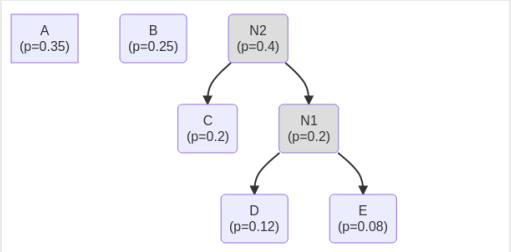
Huffman coding is the optimal coding

Finding optimal prefix-free code

Given probabilities $p_1, p_2, \ldots, p_k$ solve for code-lengths $l_i \in \mathbb{N}$, such that minimize $L(p) = \sum_{i=1}^{k} p_i l_i$

Huffman code construction

Consider a random variable X taking values in the set $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities 0.25, 0.25, 0.2, 0.15, 0.15, respectively



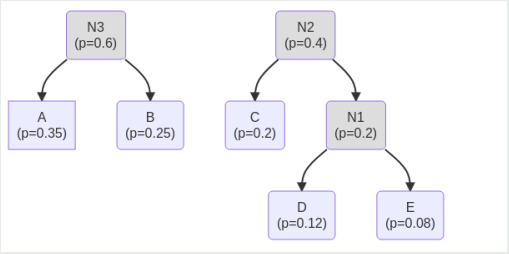
Huffman coding is the optimal coding

Finding optimal prefix-free code

Given probabilities $p_1, p_2, \ldots, p_k$ solve for code-lengths $l_i \in \mathbb{N}$, such that minimize $L(p) = \sum_{i=1}^{k} p_i l_i$

Huffman code construction

Consider a random variable X taking values in the set $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities 0.25, 0.25, 0.2, 0.15, 0.15, respectively



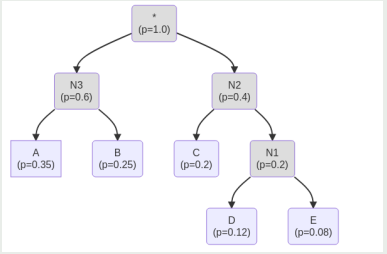
Huffman coding is the optimal coding

Finding optimal prefix-free code

Given probabilities $p_1, p_2, \ldots, p_k$ solve for code-lengths $l_i \in \mathbb{N}$, such that minimize $L(p) = \sum_{i=1}^{k} p_i l_i$

Huffman code construction

Consider a random variable X taking values in the set $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities 0.25, 0.25, 0.2, 0.15, 0.15, respectively



Huffman coding is the optimal coding

Finding optimal prefix-free code

Given probabilities $p_1, p_2, \ldots, p_k$ solve for code-lengths $l_i \in \mathbb{N}$, such that minimize $L(p) = \sum_{i=1}^{k} p_i l_i$

Huffman code construction

Consider a random variable X taking values in the set $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities 0.25, 0.25, 0.2, 0.15, 0.15, respectively
Coding table:

A	$\leftrightarrow$	00
B	$\leftrightarrow$	01
C	$\leftrightarrow$	10
D	$\leftrightarrow$	110
E	$\leftrightarrow$	111

Huffman coding is the optimal coding

Finding optimal prefix-free code

Given probabilities $p_1, p_2, \ldots, p_k$ solve for code-lengths $l_i \in \mathbb{N}$, such that minimize $L(p) = \sum_{i=1}^{k} p_i l_i$

Huffman code construction

Consider a random variable X taking values in the set $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities 0.25, 0.25, 0.2, 0.15, 0.15, respectively

Huffman coding is optimal

- see Cover and Thomas, Elements of Information Theory, Wiley

Code length

- The optimal code for coding a distribution P assigns a code to l_i of length

$$L(l_i) = -\log(p_i)$$

→ *Related to the entropy* (Shannon code)

Is it possible to propose better schemes to encode sequences?

Run-length Encoding (RLE) of a sequence

Let M be a message as a sequence of integers in $[0, 15]$, for instance

$M = \langle wwwwwwwwwwwwbwwwwwwwwwwwwbbbbwwwwwwwwwwwwwwwwwwwwwwwwbwwww$

can be represented by the sequence $\langle 12W1B12W3B24W1B14W \rangle$

Example of binary representation, concatenate:

- binary code of the sequence of repetition numbers
- binary code of the sequence letters

What we need now?

- the code sequence itself
- the code tables (for numbers and letters)
- the "program" to decode

Kolmogorov complexity [Kolmogorov 1963]

Kolmogorov Complexity is the ultimate compression

- recognizes and exploits any structure
- uncomputable, however ...

$K(s)$: Kolmogorov complexity

The Kolmogorov complexity of a binary string s is the length of the shortest program $s*$ for a universal Turing Machine U that generates s and *halts*.

$K(s \mid t)$: Conditional Complexity

The conditional Kolmogorov complexity of a string s is the length of the shortest program $s*$ for a universal Turing Machine U that given string t as input generates s and halts.

Two-part Complexity

$$K(s) = K(s') + K(s \mid s')$$

The two-part Kolmogorov complexity of a string s decomposes the shortest program $s*$ into two parts:

- length of the *algorithm*
- length of its *parameters*

$$s^*(s') = s$$

The same for encoding data (such as a databse of transactions)

- length of the *model* (include the reconstruction algorithm and code tables)
- length of its *data given model*

MDL

MDL is related to Kolmogorov Complexity

The complexity of a string is the length of the smallest program that generates the string, and then halts

Minimum Description Length (MDL)

The *Minimum Description Length* (MDL) principle given a set of models $\mathcal{M}$, the best model $M \in \mathcal{M}$ is the one that minimises

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

in which

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

Does it makes sense?

Models describe the data

- that is, they capture regularities
- hence, in an abstract way, they compress it

MDL makes this observation concrete:

the best model gives the best lossless compression

Does it makes sense?

Occam's Razor Principle

Entities should not be multiplied beyond necessity

Diagnostic parsimony: Find the fewest possible causes that explain the symptoms

Brandon has

1. cough,
2. severe abdominal pain,
3. nausea,
4. low blood pressure,
5. fever

1. pneumonia,common cold,
2. appendicitis, gout medicine,
3. food poisoning, gout medicine,
4. hemorrhage, gout medicine,
5. meningitis, common cold.

No single disease causes all of these.
Each symptom can be caused by some (possibly different) disease...
Dr. House explains the symptoms with two simple causes:

- common cold: causing the cough and fever,
- pharmacy error: cough medicine replaced by gout medicine

MDL and machine learning: the classical use

Probabilistic machine learning models are scored both on:

- Model Performance.
- Model Complexity.

MDL is a principle used in machine learning for model selection

- Akaike Information Criterion (AIC). Derived from frequentist probability.
- Bayesian Information Criterion (BIC). Derived from Bayesian probability.
- Minimum Description Length (MDL). Derived from information theory: Rissanen's MDL criterion:

$$\min_k[-\log p(s \mid m_k) + k.\log(\sqrt{n})]$$

where k is the mean squared error of the model m_k on the training sample s, n the size of s and k the number of parameters used.

How to use MDL?

To use MDL, we need to define

- define how to encode the data from a model
- compute how many bits it takes to encode a model
- compute how many bits it takes to encode the data given this model

what's a bit?

- defining an encoding $\leftrightarrow$ defining a prior
- codes and probabilities are tightly linked:
  - higher probability $\leftrightarrow$ shorter code
  - Shannon code: $L(l_i) = -log(p_i)$

MDL for itemsets mining

Let $\mathcal{I} = \{A, B, C, D, E\}$ and the following database:

t_1	$\{A, C, E\}$
t_2	$\{B\}$
t_3	$\{A, C\}$
t_4	$\{A, B, C, D\}$
t_5	$\{B, C, E\}$
t_6	$\{D\}$
t_7	$\{B, C, D, E\}$
t_8	$\{A, B, D\}$

Interesting patterns are those that compress well the patterns: i.e. **large** and **frequent** patterns!

Model of itemset mining = Code Table

Code table

- Assign a code to some itemsets patterns

Name	Pattern	Code
p_1	$\{A, C\}$	l_1
p_2	$\{B, D\}$	l_2
p_3	$\{C, E\}$	l_3
p_4	$\{A\}$	l_4
p_5	$\{B\}$	l_5
p_6	$\{C\}$	l_6
p_7	$\{D\}$	l_7
p_8	$\{E\}$	l_8

Encoding a transaction database with Code Table

Code Table:

Name	Pattern	Code
p_1	$\{A, C\}$	l_1
p_2	$\{B, D\}$	l_2
p_3	$\{C, E\}$	l_3
p_4	$\{A\}$	l_4
p_5	$\{B\}$	l_5
p_6	$\{C\}$	l_6
p_7	$\{D\}$	l_7
p_8	$\{E\}$	l_8

Transaction t: $\{B, C, E\}$

Let $\mathcal{D}$ be a database over a set of items $\mathcal{I}$, t a transaction drawn from $\mathcal{D}$, let $\mathcal{CT}$ be the set of all possible code tables over $\mathcal{I}$, and $CT \in \mathcal{CT}$ a code table. Then, $cover : \mathcal{CT} \times \mathcal{P}(I) \mapsto \mathcal{P}(\mathcal{P}(I))$ is a cover function iff it returns a set of itemsets such that

1. $cover(CT, t)$ is a subset of CT
2. if $X, Y \in cover(CT, t)$, then either $X = Y$ or $X \cap Y = \emptyset$
3. $\bigcup_{X \in cover(CT,t)} X = t$

Encoding a database: exercise

Exercise

Let us consider the following database and code table

- Find the cover of the transactions
- Updated usages of the code table
- Assign a binary code according to the Huffman code of patterns
- Estimate the length of the database encoding $L(D \mid M)$

Code Table:

Name	Pattern	Code	Usage
p_1	$\{A, C\}$	l_1	
p_2	$\{B, D\}$	l_2	
p_3	$\{C, E\}$	l_3	
p_4	$\{A\}$	l_4	
p_5	$\{B\}$	l_5	
p_6	$\{C\}$	l_6	
p_7	$\{D\}$	l_7	
p_8	$\{E\}$	l_8	

t_1	$\{A, C, E\}$
t_2	$\{B\}$
t_3	$\{A, C\}$
t_4	$\{A, B, C, D\}$
t_5	$\{B, C, E\}$
t_6	$\{D\}$
t_7	$\{B, C, D, E\}$
t_8	$\{A, B, D\}$

Encoding the model / the code table

What we need to encode the code table?

for all $p \in CT$ coded with l:

- the description of the pattern itself: $code_{ST}(p)$
- the description of the pattern code: $code_{CT}(l)$

Name	Pattern	Code
p_1	$\{A, C\}$	l_1
p_2	$\{B, D\}$	l_2
p_3	$\{C, E\}$	l_3
p_4	$\{A\}$	l_4
p_5	$\{B\}$	l_5
p_6	$\{C\}$	l_6
p_7	$\{D\}$	l_7
p_8	$\{E\}$	l_8

Description of the pattern $\{A, C\}$

- code the set $\{A, C\}$ (code the items A and C)
- code the assigned code (an integer): l_1

Standard code table (ST)

Item	Support	Code
A	$supp_{\mathcal{D}}(A)$	l_A
B	$supp_{\mathcal{D}}(B)$	l_B
C	$supp_{\mathcal{D}}(C)$	l_C
D	$supp_{\mathcal{D}}(D)$	l_D
E	$supp_{\mathcal{D}}(E)$	l_E

$$L(CT) = \sum_{c \in CT : usg(c) \neq 0} L(c \mid ST) + L(c|CT)$$

Total size of the encoding wrt to a code tabe CT

Database encoding

- for $t \in \mathcal{D}$, $$L(t \mid CT) = \sum_{X \in cover(t, CT)} L(c \mid CT)$$
- hence, $$L(\mathcal{D} \mid CT) = \sum_{t \in \mathcal{D}} L(t \mid CT)$$

Encoding of the model

$$L(CT) = \sum_{c \in CT : usg(c) \neq 0} L(c \mid ST) + L(c|CT)$$

Total size

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

J. Vreekens: "Note, we disregard *Cover* as it is identical for all CT and $\mathcal{D}$, and hence is only *a constant*" (not a complete coding of s^*)

Pattern mining with MDL?

- The encoding above enables to **compare code tables**
- if $L(\mathcal{D} \mid CT_1) < L(\mathcal{D} \mid CT_2)$ then
  - CT_1 is a better encoding
  - i.e. the itemsets of CT_1 are more interesting to represent the database

Find the optimal code table: a difficult problem?

- A coding set contains the singleton itemsets plus an almost arbitrary subset of $\mathcal{P}(I)$. Almost, since we are not allowed to choose the $|I|$ singleton itemsets.
- The number of possible coding sets is:

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

→ can not enumerate all of them ...

Algorithms

Greedy algorithms!

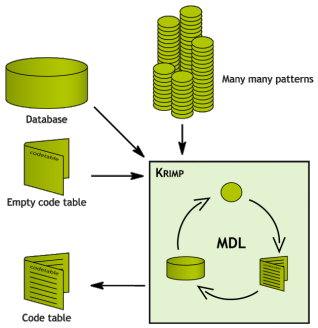
- MDL is a principle to quantitatively evaluate the interestingness of a set of patterns
- A pattern mining algorithms becomes a optimisation algorithm of the MDL criteria in the space of pattern sets $\mathcal{P}(\mathcal{I})$
- but ...
  - MDL is not good easy to optimize, and the search space is complex

  → use heuristic ... ie greedy approach

Methods to extract itemsets using MDL principle

- KRIMP
- SLIM

KRIMP: principle



Algorithm principle:

1. mine candidates from $\mathcal{D}$
2. select a subset of patterns via KRIMP
   1. iterate over candidates to select patterns to add
   2. if it improves the MDL criteria, keep it
   3. *improvement: reevaluate old elements*

KRIMP: algorithm

Algorithm 3 The KRIMP Algorithm

Input: A transaction database $\mathcal{D}$ and a candidate set $\mathcal{F}$, both over a set of items $\mathcal{I}$
Output: A heuristic solution to the Minimal Coding Set Problem, code table CT

1: $CT \leftarrow$ **Standard Code Table**($\mathcal{D}$)
2: $\mathcal{F}_o \leftarrow \mathcal{F}$ in **Standard Candidate Order** $supp_D(X)\downarrow$ $|X|\downarrow$ Lexicographically$\uparrow$
3: **for all** $F \in \mathcal{F}_o \setminus \mathcal{I}$ **do** Add the candidate one by one
4: $\quad CT_c \leftarrow (CT \cup F)$ in **Standard Cover Order**
5: $\quad$ **if** $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$ **then**
6: $\quad\quad CT \leftarrow CT_c$
7: $\quad$ **end if**
8: **end for**
9: **return** CT

KRIMP: limitation

Example

- Let us consider 3 code tables

$$CT_1 = \{\{X_1, X_2\}, \{X_1\}, \{X_2\}, \{X_3\}\}$$
$$CT_2 = \{\{X_1, X_2, X_3\}, \{X_1, X_2\}, \{X_1\}, \{X_2\}, \{X_3\}\}$$
$$CT_3 = \{\{X_1, X_2, X_3\}, \{X_1\}, \{X_2\}, \{X_3\}\}$$

- And assume that $supp_{\mathcal{D}}(\{X_1, X_2, X_3\}) = supp_{\mathcal{D}}(\{X_1, X_2\}) - 1$
- KRIMP will never consider CT_3
- But is is possible that CT_3 has a lower encoding length

Solution: pruning old elements in the code table that KRIMP

KRIMP: prune old elements

- When pruning? $\rightarrow$ just after the add of an itemset in the code table (line 6)
- Which itemsets have to be tested
  - Only the ones for which the usage has decreased
  - because their code length has increased

Algorithm 4 Code Table Post-Acceptance Pruning

Input: Code tables CT_c and CT, for a transaction database $\mathcal{D}$ over a set of items $\mathcal{I}$, where $\{X \in CT\} \subset \{Y \in CT_c\}$ and $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$.
Output: Pruned code table CT_p, such that $L(\mathcal{D}, CT_p) \leq L(\mathcal{D}, CT_c)$ and $CT_p \subseteq CT_c$.

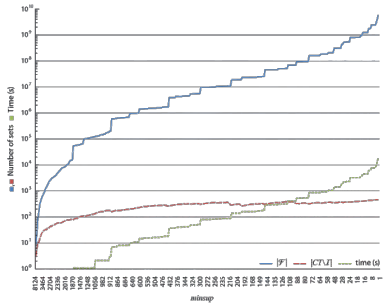
1: $PruneSet \leftarrow \{X \in CT \mid usage_{CT_c}(X) < usage_{CT}(X)\}$ Only itemsets with decreased usage
2: **while** $PruneSet \neq \emptyset$ **do**
3: $\quad PruneCand \leftarrow X \in PruneSet$ with lowest $usage_{CT_c}(X)$ usage$\downarrow$ = code length$\uparrow$
4: $\quad PruneSet \leftarrow PruneSet \setminus PruneCand$
5: $\quad CT_p \leftarrow CT_c \setminus PruneCand$
6: $\quad$ **if** $L(\mathcal{D}, CT_p) < L(\mathcal{D}, CT_c)$ **then** Compare the length of D without the itemset
7: $\quad\quad PruneSet \leftarrow PruneSet \cup \{X \in CT_p \mid usage_{CT_p}(X) < usage_{CT_c}(X)\}$
8: $\quad\quad CT_c \leftarrow CT_p$
9: $\quad$ **end if**
10: **end while**
11: **return** CT_c

KRIMP in action

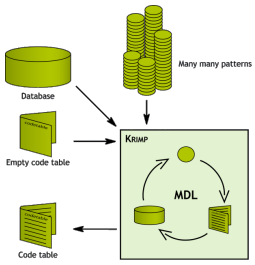
Dataset	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{F}\rvert$	KRIMP		Time
				$\lvert CT \setminus \mathcal{I}\rvert$	$\frac{L(\mathcal{D},CT)}{L(\mathcal{D},ST)}\%$	
Adult	48842	97	58461763	1303	24.4	0:02:25
Chess (kr–k)	28056	58	373421	1684	61.6	0:00:13
Led7	3200	24	15250	152	28.6	0:00:00
Letter recognition	20000	102	580968767	1780	35.7	0:52:33
Mushroom	8124	119	5574930437	442	20.6	3:40:25
Pen digits	10992	86	459191636	1247	42.3	0:31:33

- datasets the candidate set $\mathcal{F}$ was mined with $\sigma = 1$ (minsupp)
- with post-acceptance pruning
- timings with parallel implementation on quad-core 3.0 Ghz Xeon machines using four threads

KRIMP in action



So, are KRIMP code tables good?



At first glance, yes

- the code tables are characteristic in the MDL-sense
  → they compress well
- the code tables are small
  → they consist of few patterns
- the code tables are specific
  → they contain relatively long itemsets

But, are these patterns useful?

- Patterns can be evaluated on classification tasks

Compression and Classification

Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be two datasets of transactions, and CT_1/CT_2 their respective code tables.
Then, for any transaction T[a]

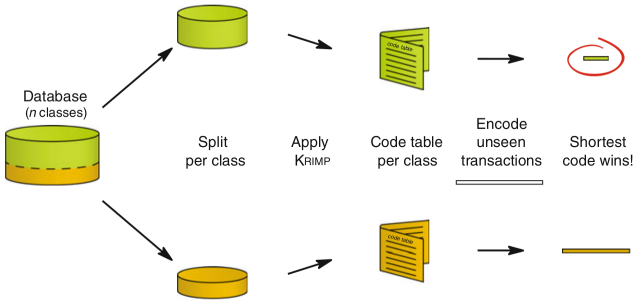
$$L(T \mid CT_1) < L(T \mid CT_2) \implies p(T \mid \mathcal{D}_\infty) > p(T \mid \mathcal{D}_\in)$$

⇒ The Bayes-optimal choice is to assign T that database that gives the best compression.

[a] optimal code: $L(c \mid CT) = -\log p(c \mid \mathcal{D})$ see article

The KRIMP Classifier

1. split database on a class attribute
2. find code tables for each databases
3. classify by compression



The KRIMP Classifier: results

Dataset	KRIMP	C4.5	CBA	HRM	iCAEP	SVM
Adult	84.3	**85.5**	84.2	81.9	80.9	84.7
Anneal	96.6	**97.8**	94.7		95.1	95.3
Breast	94.1	94.1	94.0		**97.4**	93.7
Chess (k–k)	90.0	**99.4**	72.8		94.6	93.9
Chess (kr–k)	57.9	**78.5**	25.8[a]	44.9		46.3
Connect–4	69.4	**80.1**	68.7	68.1	69.9	77.6
Heart	61.7	54.8	57.3		**80.3**	58.4
Ionosphere	**91.0**	90.9	87.2[b]		90.6	90.9
Iris	**96.0**	94.0	94.0	94.7	93.3	**94.0**
Led7	75.3	75.3	66.6	74.6		**75.8**
Letter	70.9	**77.5**	28.6	76.8		69.8
Mushroom	**100**	**100**	46.4	99.9	99.8	99.9
Nursery	92.3	**99.5**	90.1	92.8	84.7	97.6
Page blocks	**92.6**	92.5	90.9	91.6		92.2
Pen digits	95.0	95.6	87.4	96.2		**96.6**
Pima	72.7	72.7	**75.0**	73.0	72.3	74.0
Tic–tac–toe	88.7	93.3	**100**	81.0	92.1	87.9
Waveform	77.1	74.1	77.6	80.5	**81.7**	80.1
Wine	**100**	96.6	53.2	63.0	98.9	97.2
Average	84.5	87.0	75.4			84.5

MDL for sequences

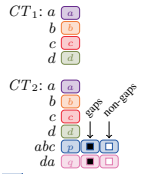
How to adapt MDL for sequence databases

$$T = \langle a, b, d, c, a, d, b, a, a, b, c \rangle$$

- Same principles as KRIMP
  - define an encoding from collection of sequential patterns
  - evaluate the coding length
- Difficulties: gaps
- SQS method (read "squeeze")

SQS encoding

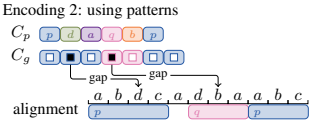
- **Code table**
  - 4 columns
    - Pattern,
    - code,
    - gap code,
    - no gap code



- **Encoded database = 2 streams**
  - Cp: pattern stream
  - Cg: gap stream

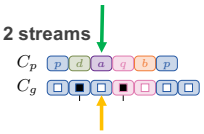
Encoding 1: using only singletons

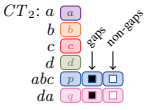
Encoding 2: using patterns

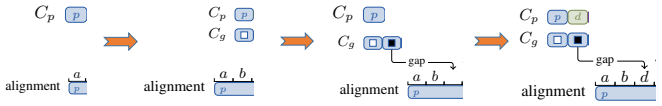


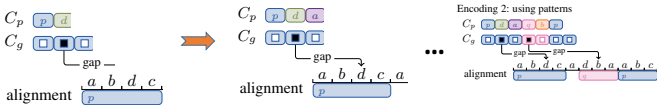
SQS encoding

- **Encoded database = 2 streams**
  - Cp: pattern stream
  - Cg: gap stream



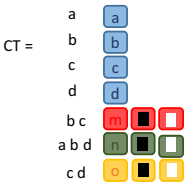


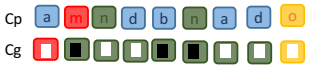




SQS encoding: exercice

- **Let us consider the following code table and the following streams**
- **Give the decoded sequence**





SQS coding length

Encoding length

- Patterns

$$L(code_p(X) \mid CT) = -\log\left(\frac{usage(X)}{\sum_{Y \in CT} usage(Y)}\right)$$

- Let
  - *gaps*(*X*) = number of gaps in the usage of pattern X
  - *fills*(*X*) = number of non-gaps in the usage of pattern X
    - Note: $fills(X) = usage(X)(|X| - 1)$
- Gaps

$$L(code_g(X) \mid CT) = -\log\left(\frac{gaps(X)}{gaps(X) + fills(X)}\right)$$

- Non-gaps

$$L(code_n(X) \mid CT) = -\log\left(\frac{fills(X)}{gaps(X) + fills(X)}\right)$$

SQS coding length

Encoding length

- Pattern stream

$$L(C_p \mid CT) = \sum_{X \in CT} usage(X) L(code_p(X))$$

- Gap stream

$$L(C_g \mid CT) = \sum_{\substack{X \in CT \\ |X|>1}} \Big(gaps(X) L(code_g(X)) + fills(X) L(code_n(X))\Big)$$

- Database D with a code table CT

$$L(D \mid CT) = L_{\mathbb{N}}(|D|) + \sum_{S \in D} L_{\mathbb{N}}(|S|) + L(C_p \mid CT) + L(C_g \mid CT)$$

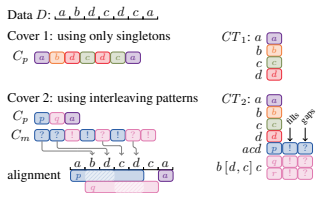
Number of sequences in D

Length of a sequence S in D

Improvement of SQS: SQUISH

Limitation of SQS

- only sequences of items: what if sequences of itemsets?
- no interleaving patterns: two occurrences of patterns can not overlaps?
  → SQUISH [BV17] proposes an encoding



What's about s^*?

Remind that the program to decode is not take into account into the length ...

- can I compare the coding length of SQS and the length of SQUISH?
- ... but s^* contains a lot of assumptions

Conclusions

- MDL is great for picking important and useful patterns
- KRIMP approximates the MDL ideal very well
  - vast reduction of the number of itemsets
  - works for other pattern types equally well: itemsets, sequences, streams
- Local patterns and information theory
  - naturally induce good classifiers, clusterers, distance measures
  - with instant characterisation and explanation,
  - and, without (explicit) parameters

Conclusions

When using MDL for pattern mining

- Question 1: what kind of model to use?
  → Data modelisation
- Question 2: how to encode the database? the model?
  → Information theory
- Question 3: how to find the "best" (or at least one good) model?
  → Algorithmic (heuristic)

→ lot of assumptions hidden behind the encoding and in the heuristic solving ...
→ maybe also a kind of declarative programming

Some additional references

- [VVLS11] Krimp: mining itemsets that compress, Jilles Vreeken, Matthijs van Leeuwen, Arno Siebes, DMKD 2011
- [SV12] Slim: Directly Mining Descriptive Patterns, Koen Smets and Jilles Vreeken, SDM 2012
- [TV12] The Long and the Short of It: Summarising Event Sequences with Serial Episodes, Nikolaj Tatti and Jilles Vreeken, KDD 2012
- [Grü07] The Minimum Description Length Principle, Peter Grünwald, MIT Press, 2007
- [Gal22] The Minimum Description Length Principle for Pattern Mining, A Survey, Esther Galbrun, DMKD, 2022

Outline

Pattern mining

The generic pattern mining problem can be rewritten as finding:

$$\{X \in \mathcal{L} \mid \mathcal{Q}(X, D)\}$$

where

- D is a dataset of examples
- $(\mathcal{L}, \lhd)$ is a pattern language (a poset)
- $\mathcal{Q}(X, D)$ is a *constraint*

where $\mathcal{Q}(\cdot, D)$ is (anti-)monotone w.r.t. $\lhd$.

Other tasks

- rules
- maximal patterns
- closed patterns

Pattern mining

A wide range of pattern domains

- itemsets in database relation
- numerical databases
- sequences, temporal patterns, periodic patterns
- graphs, ...

One main anti-monotone constraints: the support

- support: interesting measure but incomplete
- based on the containment relation of a pattern in a object
  - holds the semantics of the pattern
- not always easy to define nor to evaluate
  - different counting of occurrences in sequences or graphs
  - evaluate the number of occurrence can be complex (subgraph isomorphism)
- alternative measures: surface, lift, ...

Pattern mining

Pattern mining is a combinatorial task

- pattern domain = search space
- pattern mining task = definition of constraints on pattern
- challenge 1 : efficiently traverse the search space to find the "good" patterns

The "a priori" trick

- Use constraint anti-monotonie: $p \lhd q \implies (c(q) \Rightarrow c(p))$

2 decades of algorithm developments

- improve efficiency of the search space by fast pruning
- prevent redundant pattern exploration
- ... but lead to overwhelming number of patterns!

Pattern mining

Challenge 2 : reduce the number of patterns

- add more and flexible constraints ⇝ constraint programming approaches
- use statistically representatives
  - ⇝ sampling techniques
  - ⇝ MCTS methods
- add less but meaningfull patterns ⇝ MDL approaches

Thanks for listening

Questions ?

References I

Apratim Bhattacharyya and Jilles Vreeken, *Efficiently summarising event sequences with rich interleaving patterns*, Proceedings of the 2017 SIAM International Conference on Data Mining, SIAM, 2017, pp. 795–803.

Esther Galbrun, *The minimum description length principle for pattern mining: A survey*, Data mining and knowledge discovery 36 (2022), no. 5, 1679–1727.

Martin Gebser, Thomas Guyet, René Quiniou, Javier Romero, and Torsten Schaub, *Knowledge-based sequence mining with ASP*, Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, 2016, pp. 1497–1504.

Tias Guns, Siegfried Nijssen, and Luc De Raedt, *Itemset mining: A constraint programming perspective*, Artificial Intelligence 175 (2011), no. 12-13, 1951–1983.

Peter D Grünwald, *The minimum description length principle*, MIT press, 2007.

Matti Järvisalo, *Itemset mining as a challenge application for answer set enumeration*, Logic Programming and Nonmonotonic Reasoning, Springer, 2011, pp. 304–310.

Benjamin Negrevergne and Tias Guns, *Constraint-based sequence mining using constraint programming*, Proceedings of International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR, 2015, pp. 288–305.

Koen Smets and Jilles Vreeken, *Slim: Directly mining descriptive patterns*, Proceedings of the 2012 SIAM international conference on data mining, SIAM, 2012, pp. 236–247.

References II

Nikolaj Tatti and Jilles Vreeken, *The long and the short of it: summarising event sequences with serial episodes*, Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012, pp. 462–470.

Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes, *Krimp: mining itemsets that compress*, Data Mining and Knowledge Discovery 23 (2011), 169–214.

Apratim Bhattacharyya and Jilles Vreeken, *Efficiently summarising event sequences with rich interleaving patterns*, Proceedings of the 2017 SIAM International Conference on Data Mining, SIAM, 2017, pp. 795–803.

Esther Galbrun, *The minimum description length principle for pattern mining: A survey*, Data mining and knowledge discovery 36 (2022), no. 5, 1679–1727.

Martin Gebser, Thomas Guyet, René Quiniou, Javier Romero, and Torsten Schaub, *Knowledge-based sequence mining with ASP*, Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, 2016, pp. 1497–1504.

Tias Guns, Siegfried Nijssen, and Luc De Raedt, *Itemset mining: A constraint programming perspective*, Artificial Intelligence 175 (2011), no. 12-13, 1951–1983.

Peter D Grünwald, *The minimum description length principle*, MIT press, 2007.

Matti Järvisalo, *Itemset mining as a challenge application for answer set enumeration*, Logic Programming and Nonmonotonic Reasoning, Springer, 2011, pp. 304–310.

References III


Benjamin Negrevergne and Tias Guns, *Constraint-based sequence mining using constraint programming*, Proceedings of International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR, 2015, pp. 288–305.


Koen Smets and Jilles Vreeken, *Slim: Directly mining descriptive patterns*, Proceedings of the 2012 SIAM international conference on data mining, SIAM, 2012, pp. 236–247.


Nikolaj Tatti and Jilles Vreeken, *The long and the short of it: summarising event sequences with serial episodes*, Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining, 2012, pp. 462–470.


Jilles Vreeken, Matthijs Van Leeuwen, and Arno Siebes, *Krimp: mining itemsets that compress*, Data Mining and Knowledge Discovery 23 (2011), 169–214.