SLIM: Directly Mining Descriptive Patterns

Koen Smets Jilles Vreeken

SDM12

Pattern mining

- Discovering all patterns
  - Fulfilling constraints
  - Potentially interesting

Pattern mining has drawbacks

- Discovering all patterns
  - Fulfilling constraints, but too many
  - Potentially interesting, but redundant

~~Pattern mining~~ has drawbacks

Pattern set mining

- Discovering all patterns
  - Fulfilling constraints, but too many
  - Potentially interesting, but redundant
- Discovering high-quality set of patterns
  - Small
  - Useful

~~Pattern mining~~ has drawbacks

Pattern set mining

- Discovering all patterns
  - Fulfilling constraints, but too many
  - Potentially interesting, but redundant
- Discovering high-quality set of patterns
  - Small
  - Useful
- Identify the best set of patterns
  - Together describe the data best

MDL approach to pattern set mining

(Siebes et al 2006 / Vreeken et al 2011)

MDL approach to pattern set mining

(Siebes et al 2006 / Vreeken et al 2011)

Minimum Description Length principle

Given a set of models $\mathcal{M}$, the best model is

$$\underset{M \in \mathcal{M}}{\operatorname{argmin}}\ L(M) + L(\mathcal{D} \mid M)$$

MDL approach to pattern set mining

(Siebes et al 2006 / Vreeken et al 2011)

Minimum Description Length principle

Given a set of models $\mathcal{M}$, the best model is

$$\underset{M \in \mathcal{M}}{\operatorname{argmin}}\ L(M) + L(\mathcal{D} \mid M)$$

MDL for pattern set mining

- Model = set of patterns
- Lossless compression of the data

MDL approach to pattern set mining

(Siebes et al 2006 / Vreeken et al 2011)

Minimum Description Length principle

Given a set of models $\mathcal{M}$, the best model is

$$\underset{M \in \mathcal{M}}{\operatorname{argmin}}\ L(M) + L(\mathcal{D} \mid M)$$

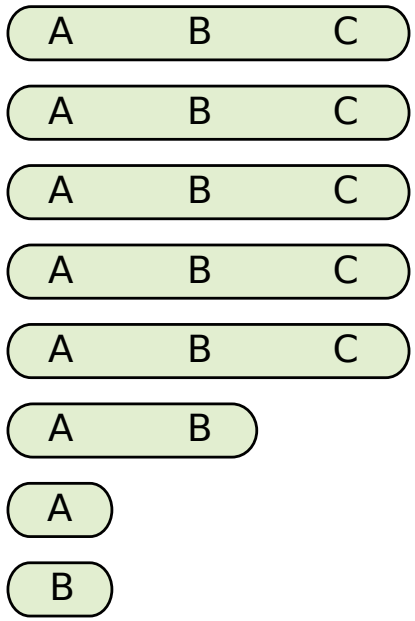
MDL for pattern set mining

- Model = set of patterns
- Lossless compression of the data
- Properties
  - Non-redundant
  - Not overly simple
  - Not overly complex

MDL approach to pattern set mining

(Siebes et al 2006 / Vreeken et al 2011)

Transaction database



MDL approach to pattern set mining

(Siebes et al 2006 / Vreeken et al 2011)

Code table

Itemset	*Code*	*Usage*
A B C	0.85bits	5
A	2.17bits	2
B	2.17bits	2
C	—	0

Transaction database

A B C

A B C

A B C

A B C

A B C

A B

A

B

MDL approach to pattern set mining

(Siebes et al 2006 / Vreeken et al 2011)

Code table

Itemset	*Code*	*Usage*
A B C	0.85bits	5
A	2.17bits	2
B	2.17bits	2
C	—	0

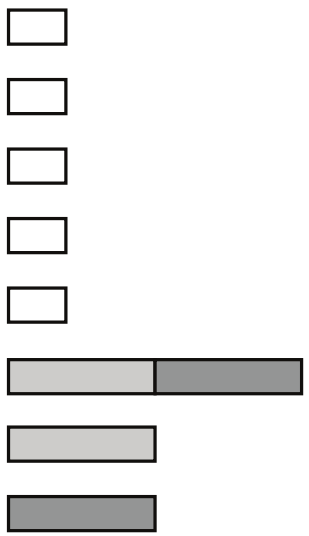
Transaction database

A B C
A B C
A B C
A B C
A B C
A B
A
B

Covered database

A B C
A B C
A B C
A B C
A B C
A B
A
B

Encoded database



How to mine the optimal code table?

How to mine the optimal code table?

- Easier said than done

How to mine the optimal code table?

- Easier said than done
- The number of possible code tables is huge
  - Exponential in the number of candidate itemsets

How to mine the optimal code table?

- Easier said than done

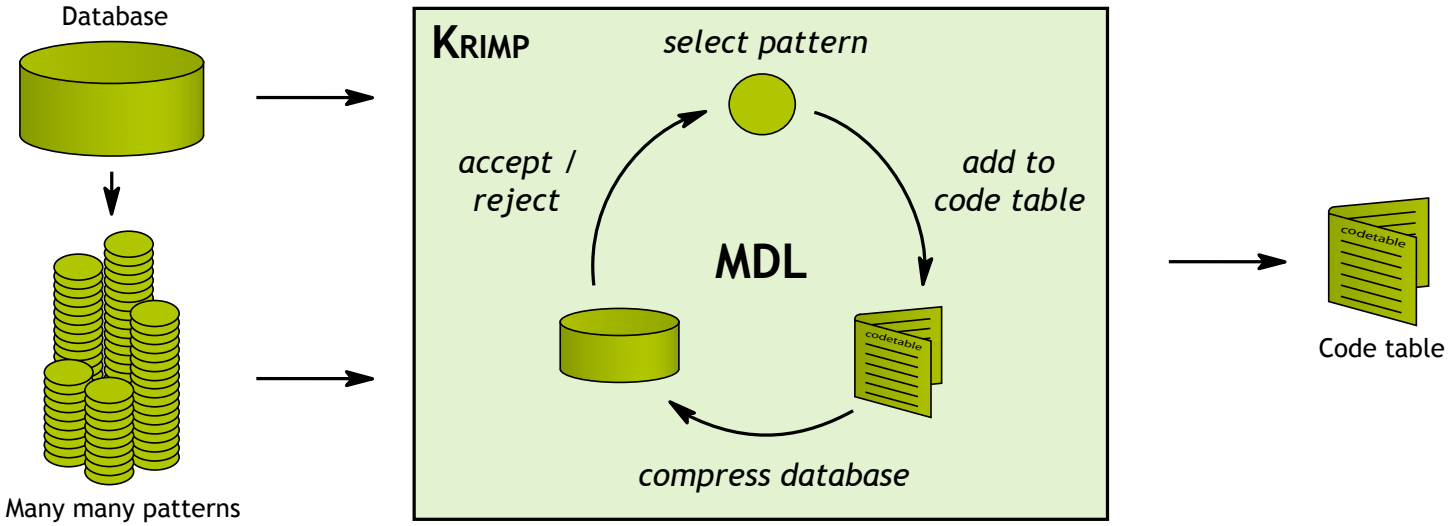
- The number of possible code tables is huge
  - Exponential in the number of candidate itemsets
- No useful structure to exploit

How to mine the optimal code table?

- Easier said than done
- The number of possible code tables is huge
  - Exponential in the number of candidate itemsets
- No useful structure to exploit
- Hence, we resort to heuristics

KRIMP

(Siebes et al 2006 / Vreeken et al 2011)



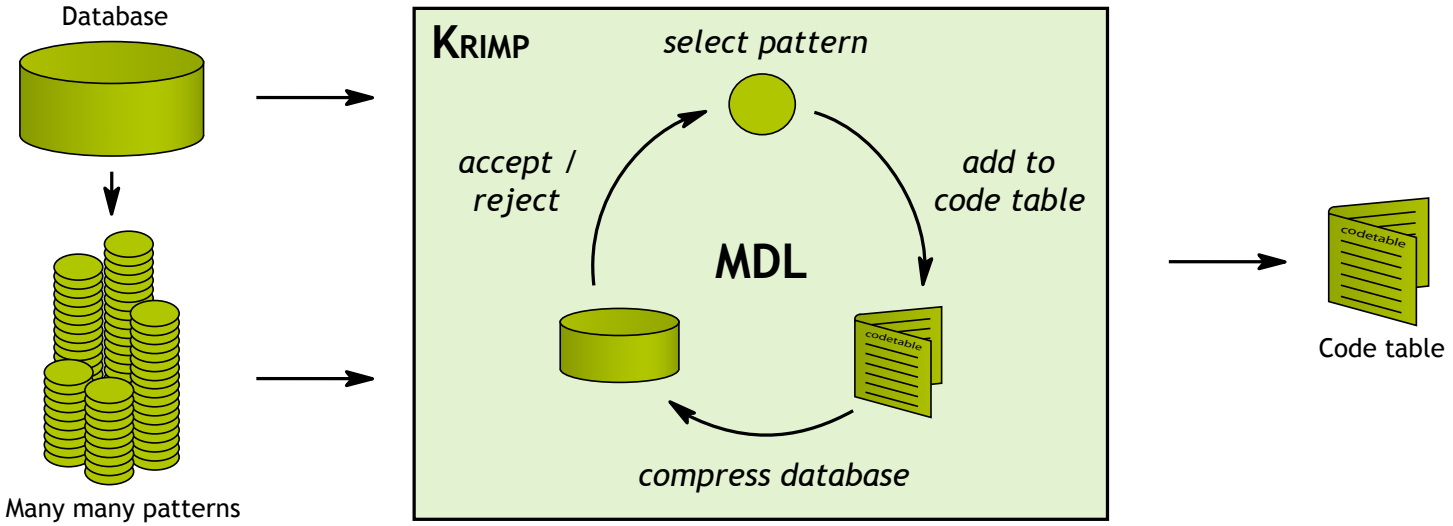
Standard two phase approach

1. Mining candidate patterns

2. Considering candidates once in a fixed order

KRIMP

(Siebes et al 2006 / Vreeken et al 2011)



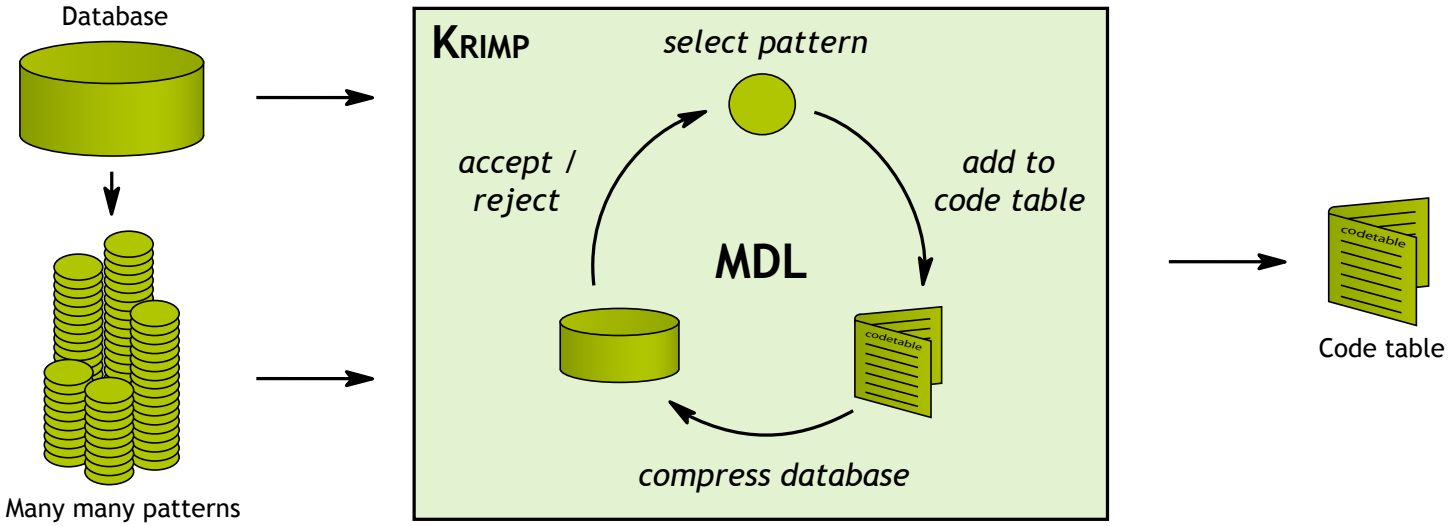
Standard two phase approach has drawbacks

1. Mining candidate patterns

2. Considering candidates once in a fixed order

KRIMP

(Siebes et al 2006 / Vreeken et al 2011)



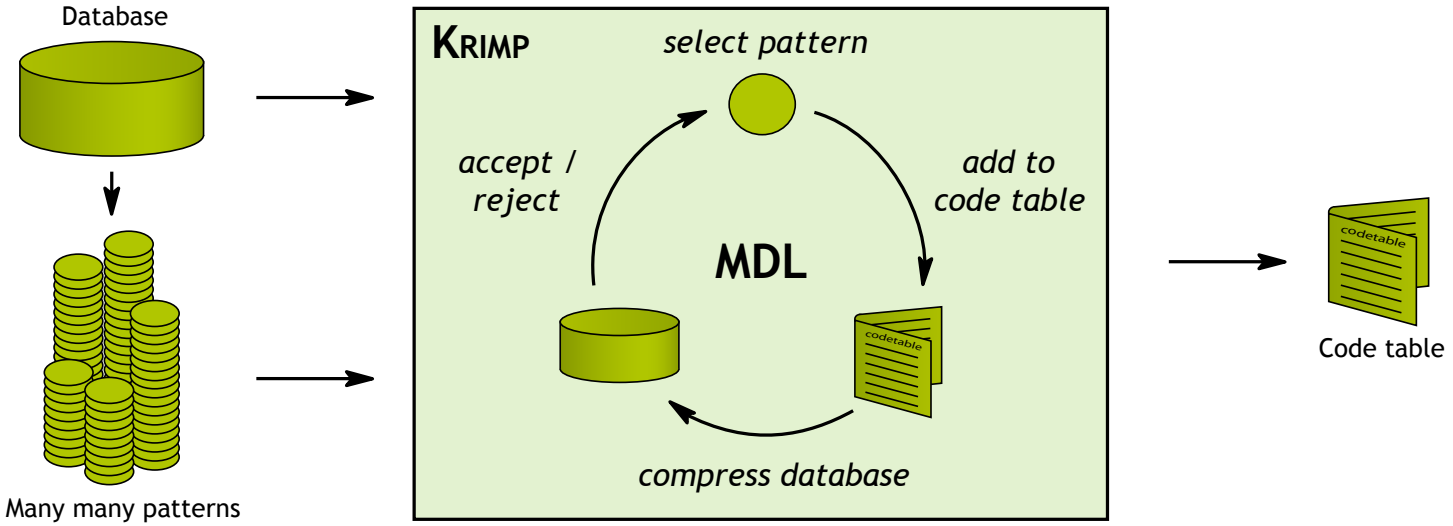
Standard two phase approach has drawbacks

1. Mining candidate patterns is expensive

2. Considering candidates once in a fixed order

KRIMP

(Siebes et al 2006 / Vreeken et al 2011)



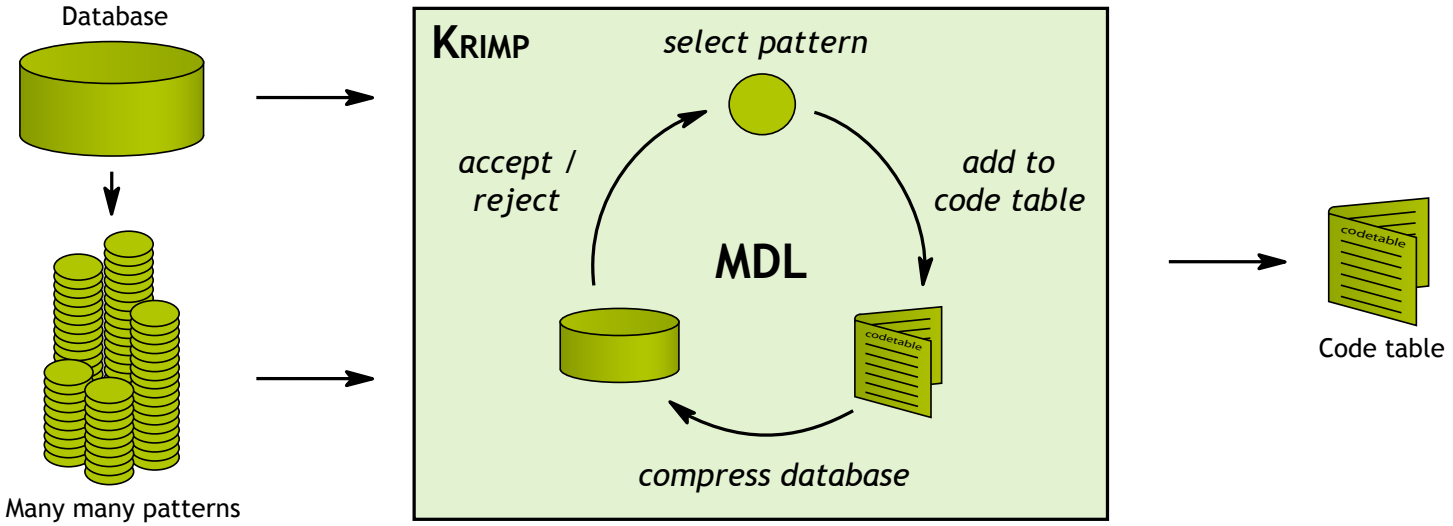
Standard two phase approach has drawbacks

1. Mining candidate patterns is expensive
   - Lower support threshold correspond to better results

2. Considering candidates once in a fixed order

KRIMP

(Siebes et al 2006 / Vreeken et al 2011)



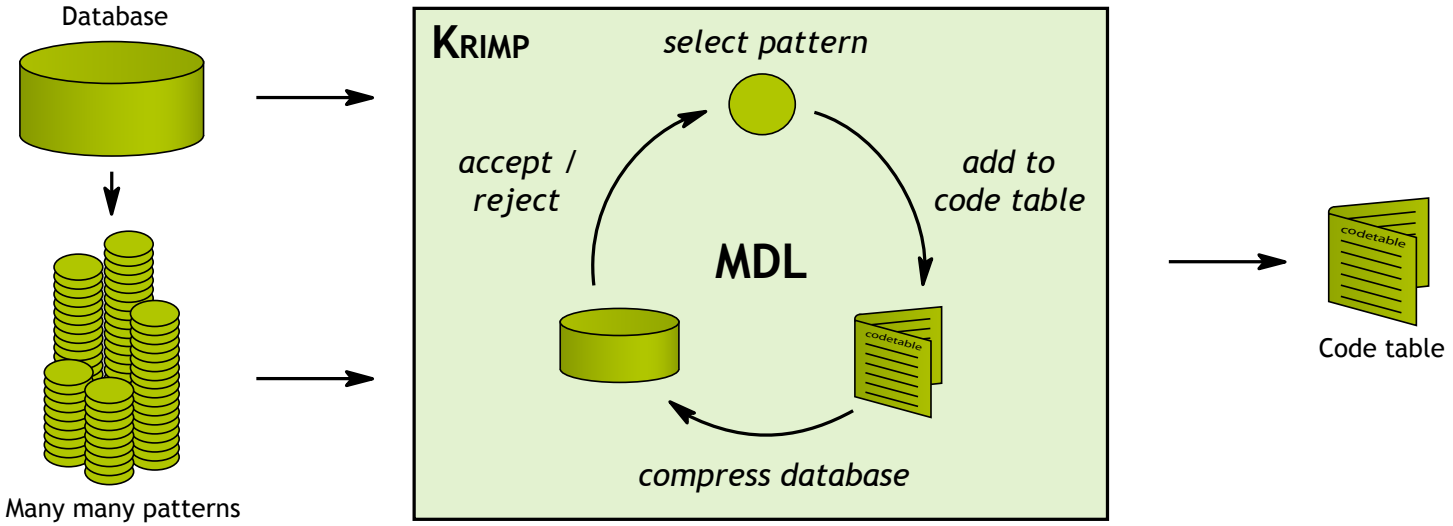
Standard two phase approach has drawbacks

1. Mining candidate patterns is expensive
   - Lower support threshold correspond to better results
   - Pattern explosion prohibits detailed analysis

2. Considering candidates once in a fixed order

KRIMP

(Siebes et al 2006 / Vreeken et al 2011)

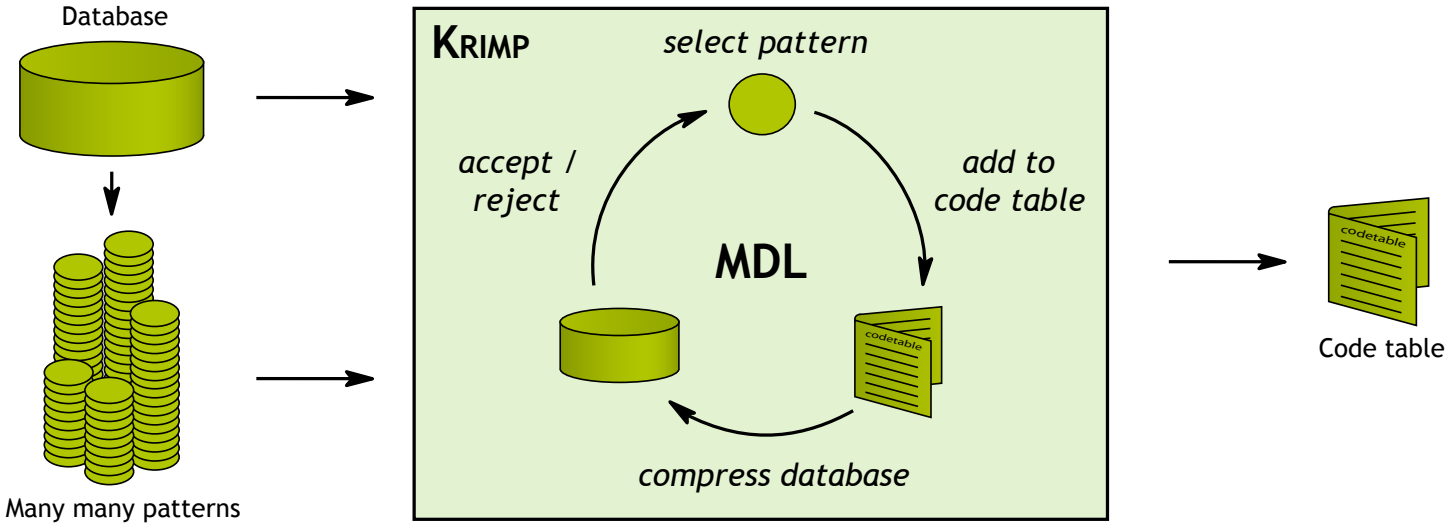


Standard two phase approach has drawbacks

1. Mining candidate patterns is expensive
   - Lower support threshold correspond to better results
   - Pattern explosion prohibits detailed analysis
   - Most candidates will be rejected
2. Considering candidates once in a fixed order

KRIMP

(Siebes et al 2006 / Vreeken et al 2011)

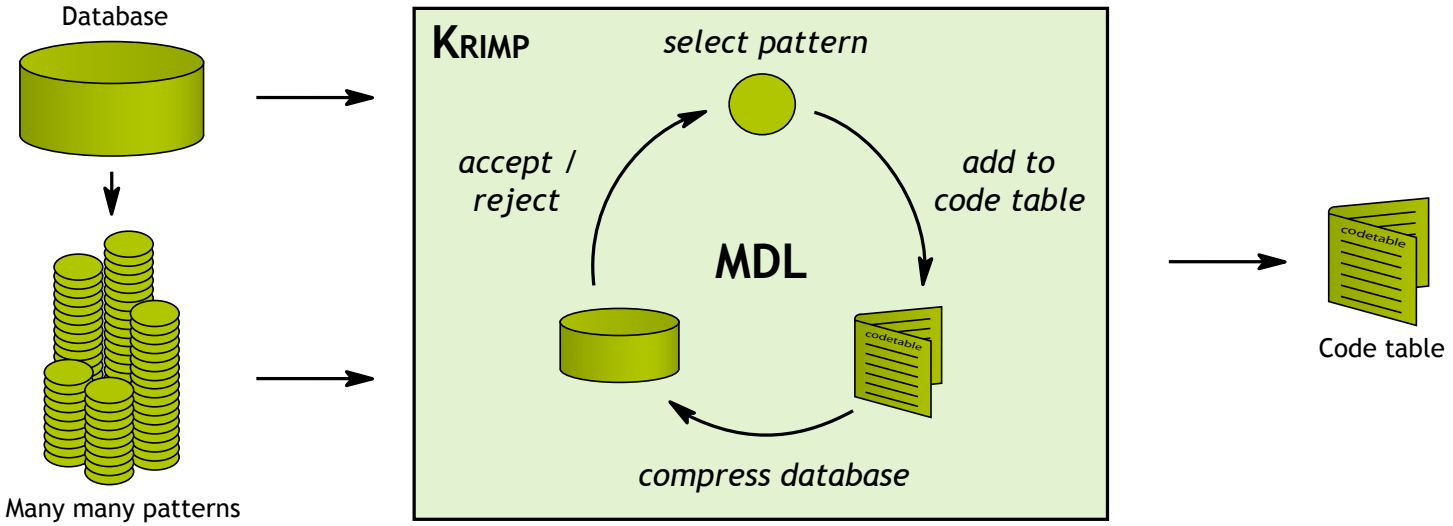


Standard two phase approach has drawbacks

1. Mining candidate patterns is expensive
   - Lower support threshold correspond to better results
   - Pattern explosion prohibits detailed analysis
   - Most candidates will be rejected
2. Considering candidates once in a fixed order is suboptimal

KRIMP

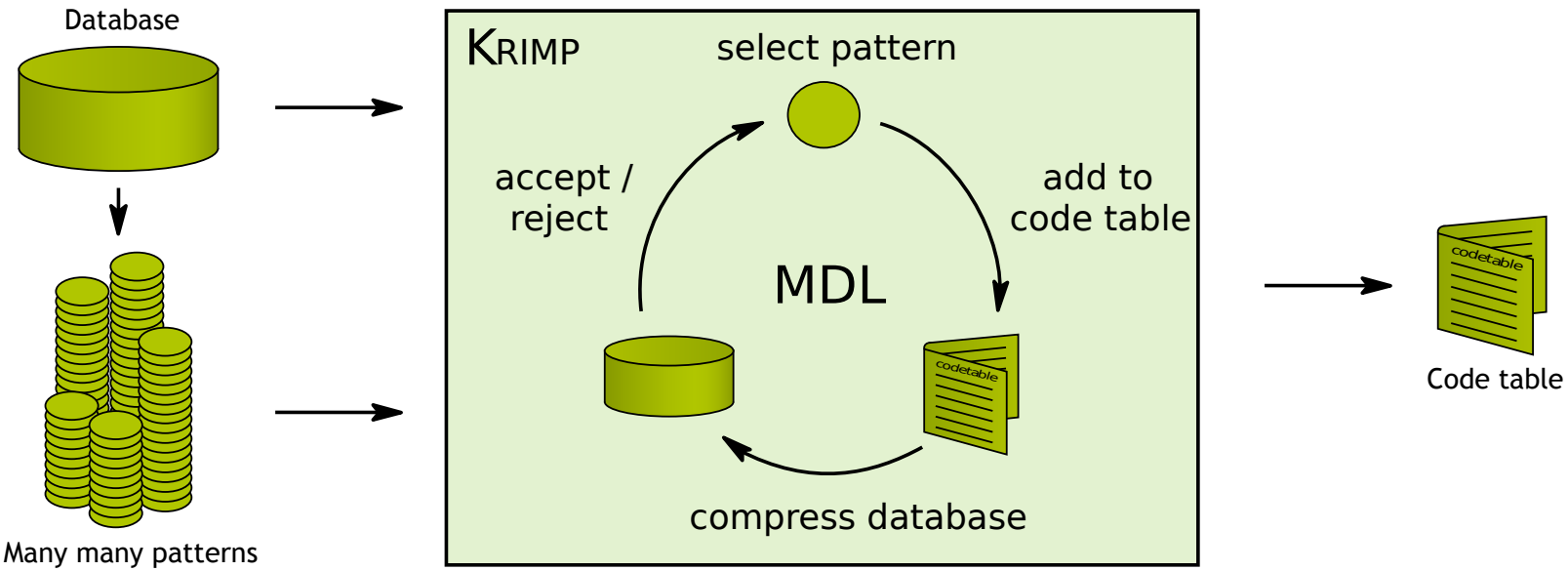
(Siebes et al 2006 / Vreeken et al 2011)



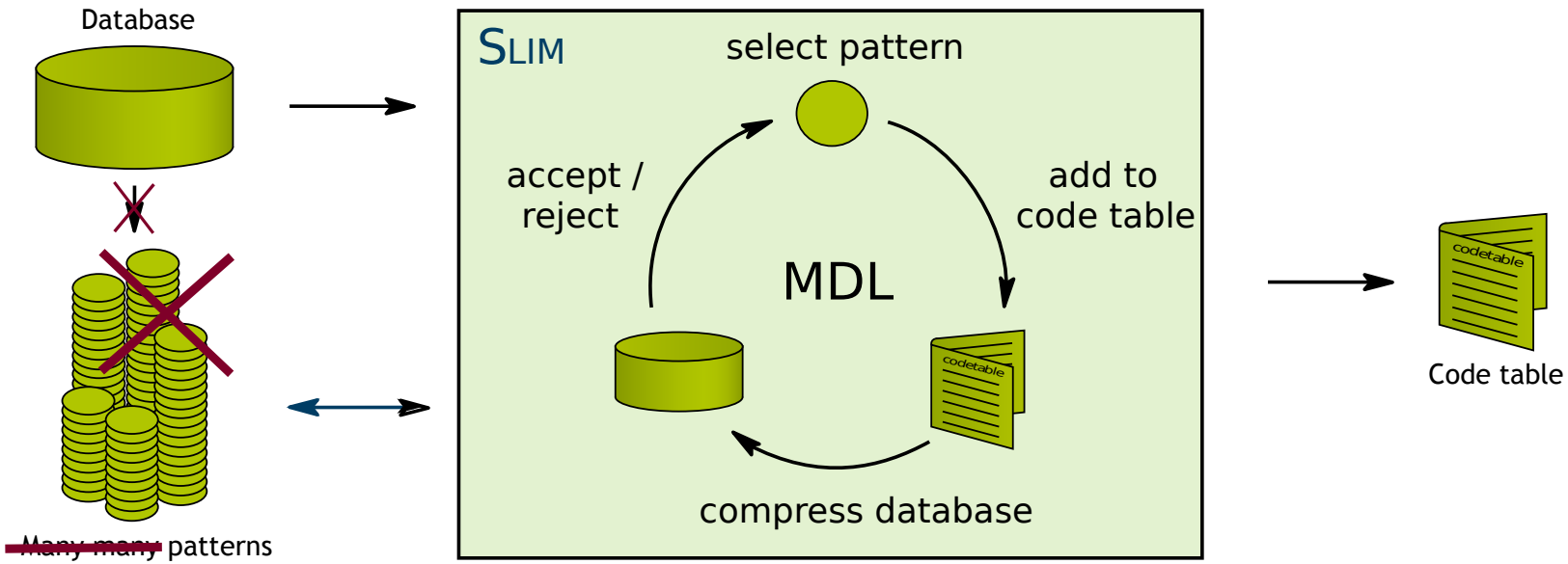
Standard two phase approach has drawbacks

1. Mining candidate patterns is expensive
   - Lower support threshold correspond to better results
   - Pattern explosion prohibits detailed analysis
   - Most candidates will be rejected
2. Considering candidates once in a fixed order is suboptimal
   - Rejecting candidates that could be useful later on

KRIMP → SLIM


Database
Many many patterns
KRIMP
select pattern
accept / reject
add to code table
MDL
compress database
codetable
codetable
Code table

KRIMP → SLIM


Database
Many-many patterns
SLIM
select pattern
accept / reject
add to code table
MDL
compress database
codetable
codetable
Code table

Maximise compression locally

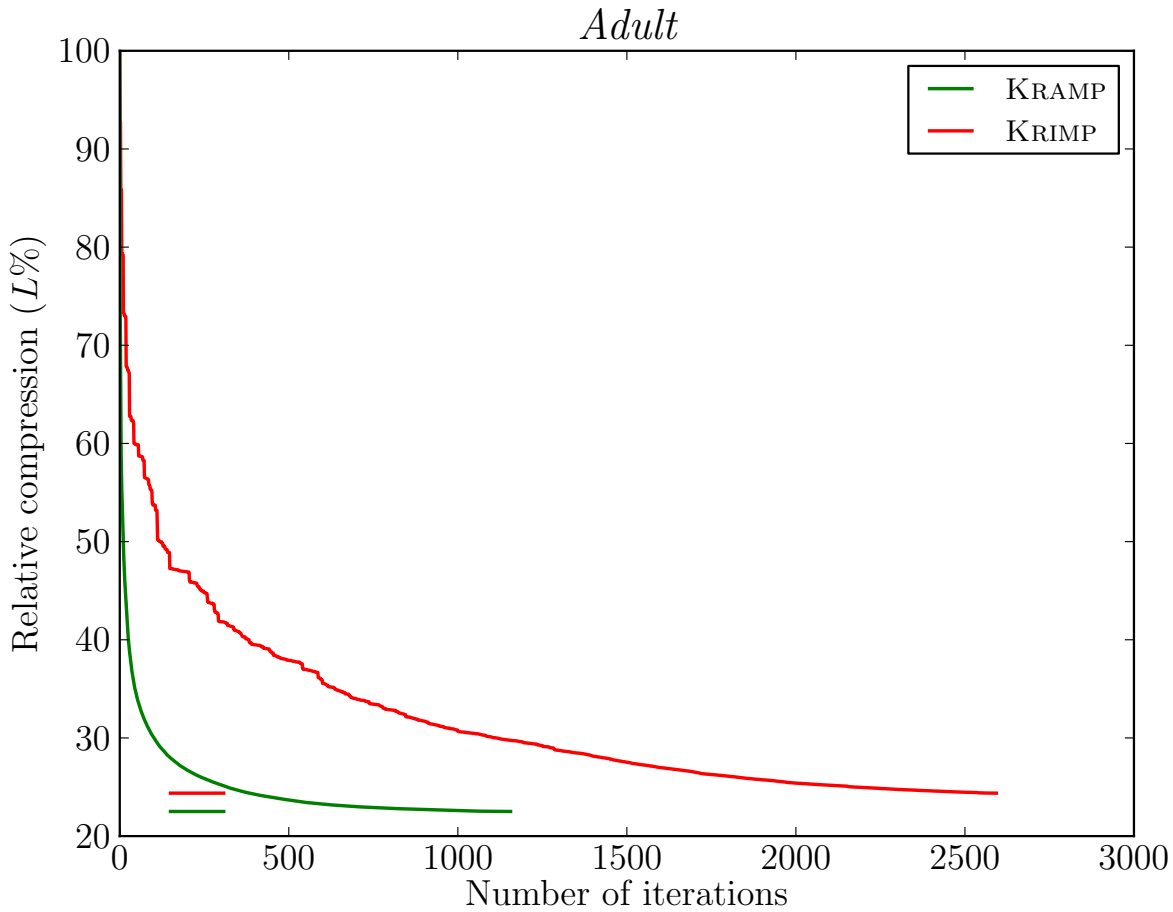
- Select the best addition out of all candidates

Maximise compression locally

- Select the best addition out of all candidates
- Converging quickly to better compression

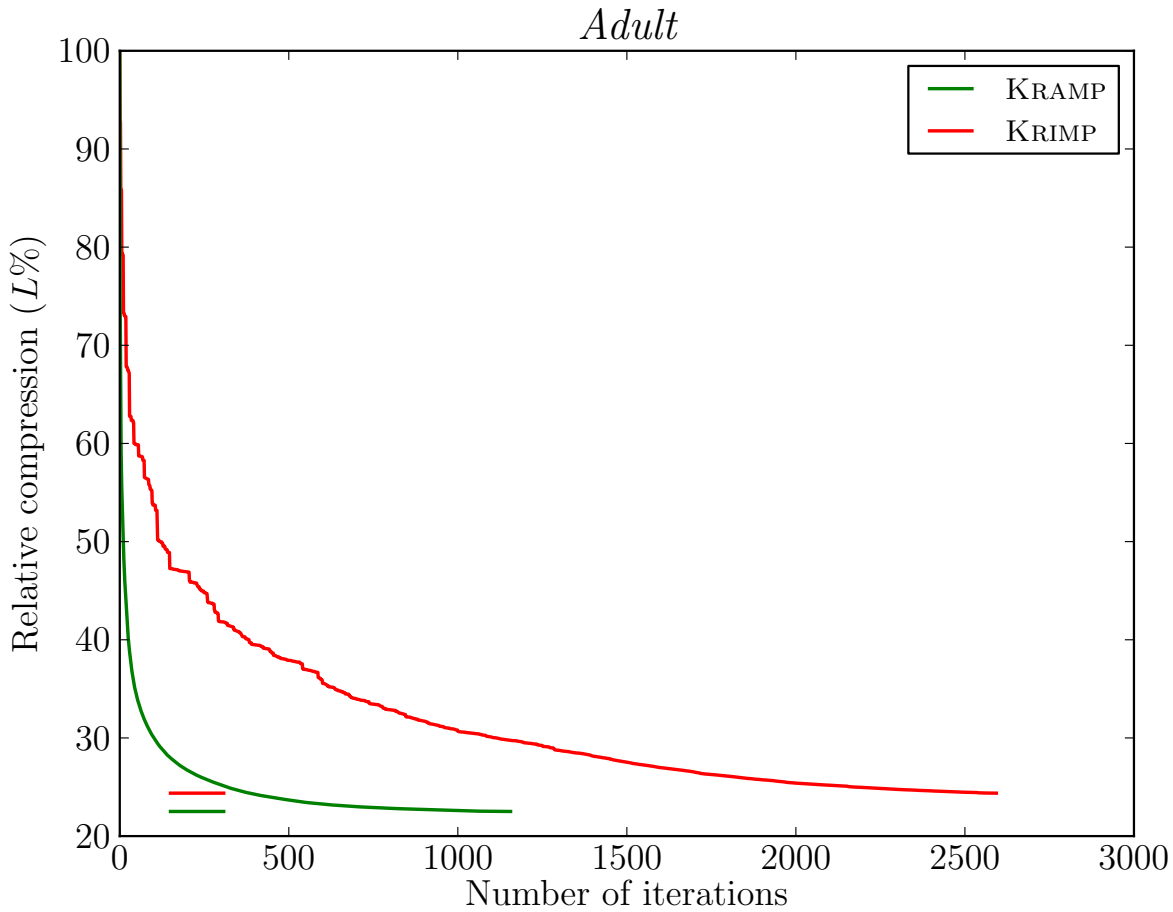
Maximise compression locally

- Select the best addition out of all candidates
- Converging quickly to better compression



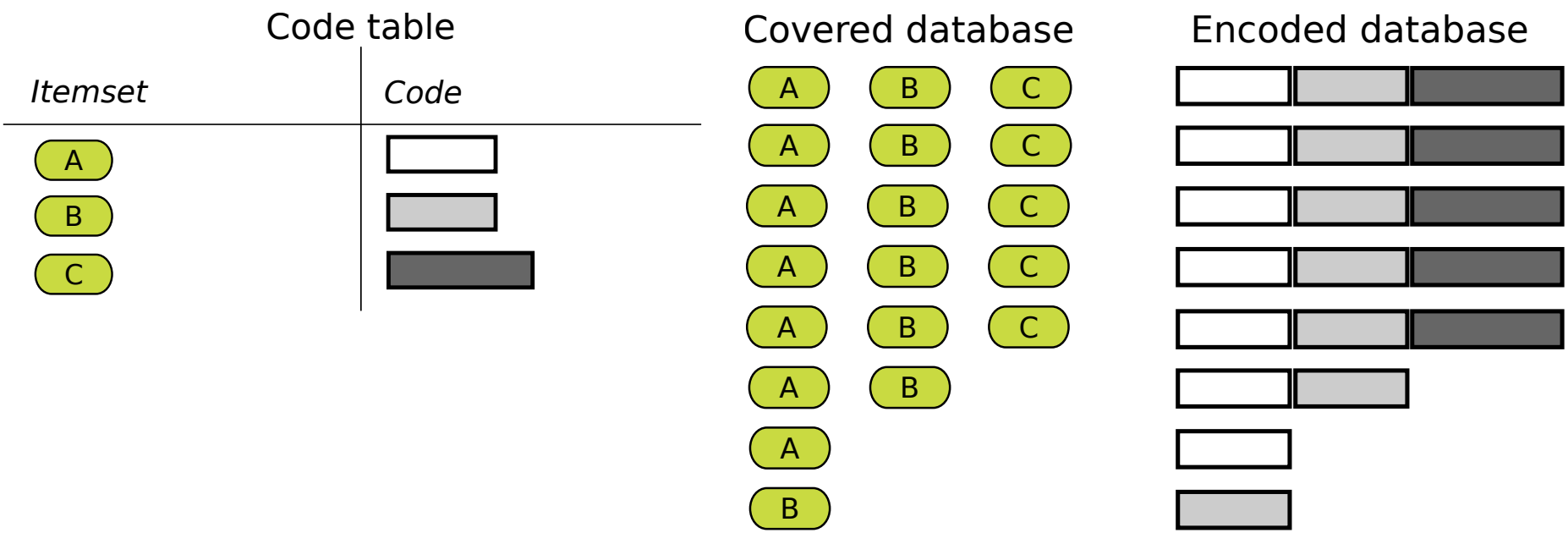
Maximise compression locally

- Select the best addition out of all candidates is infeasible
  - Reordering all candidates means recompressing database
- Converging ~~quickly~~ to better compression takes 2 months



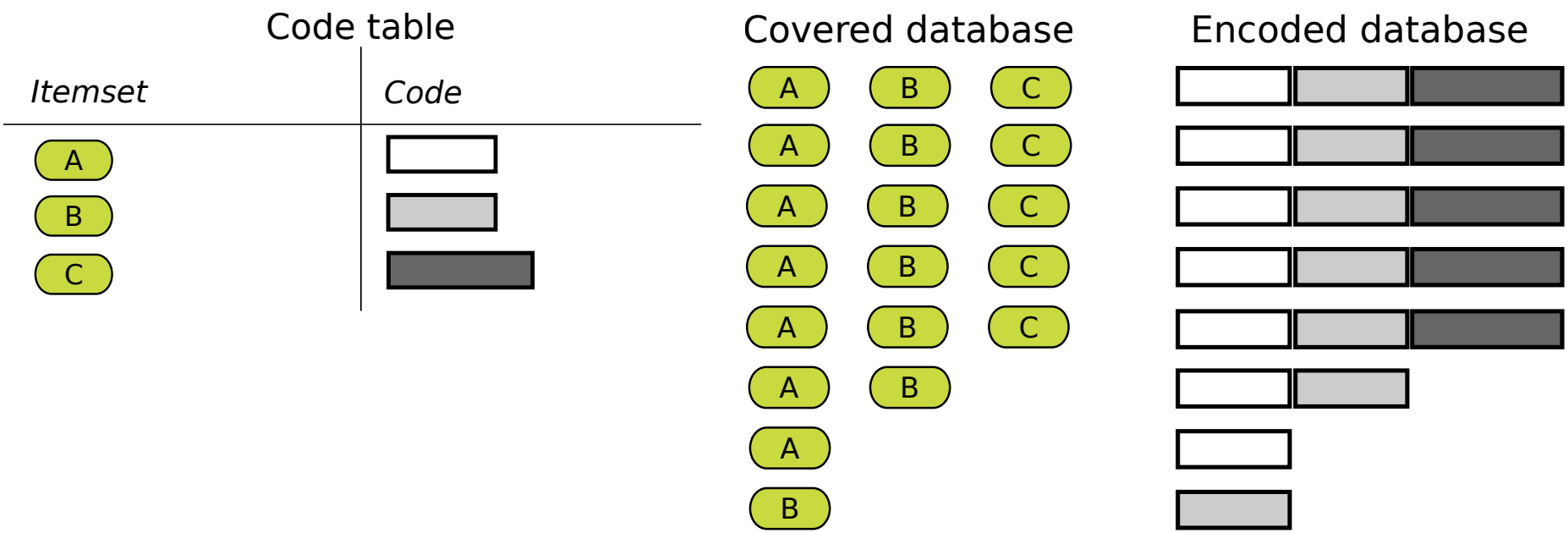
Greedily construct code table bottom-up

- Start with code table containing only singletons



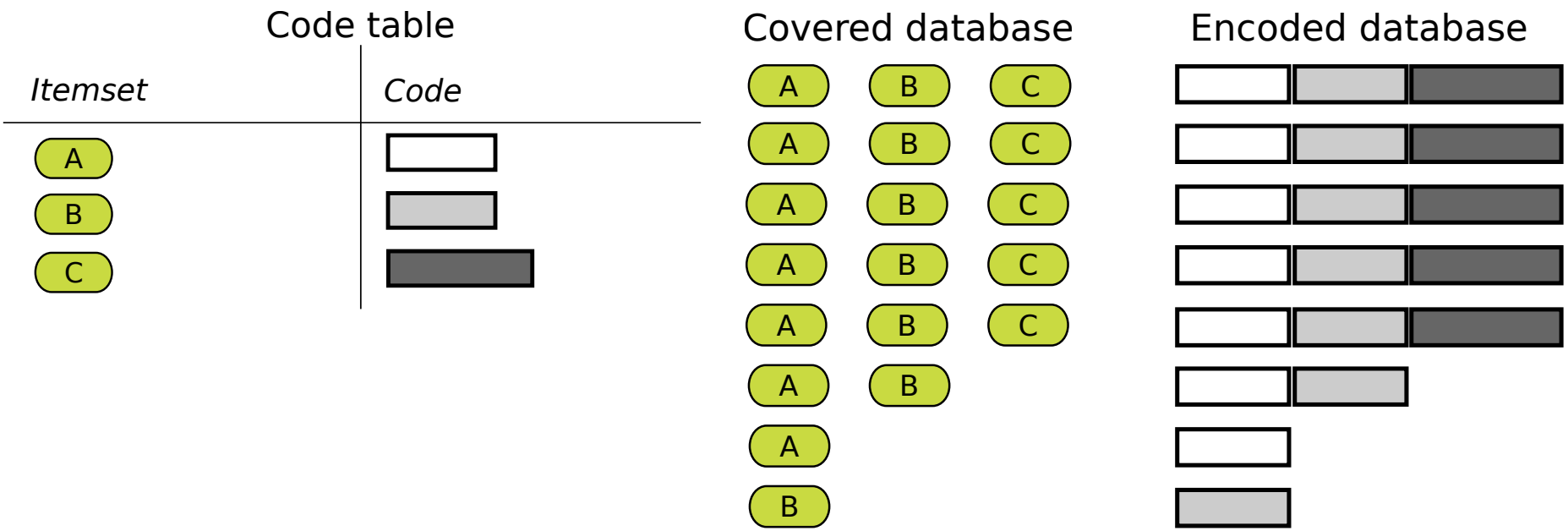
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally



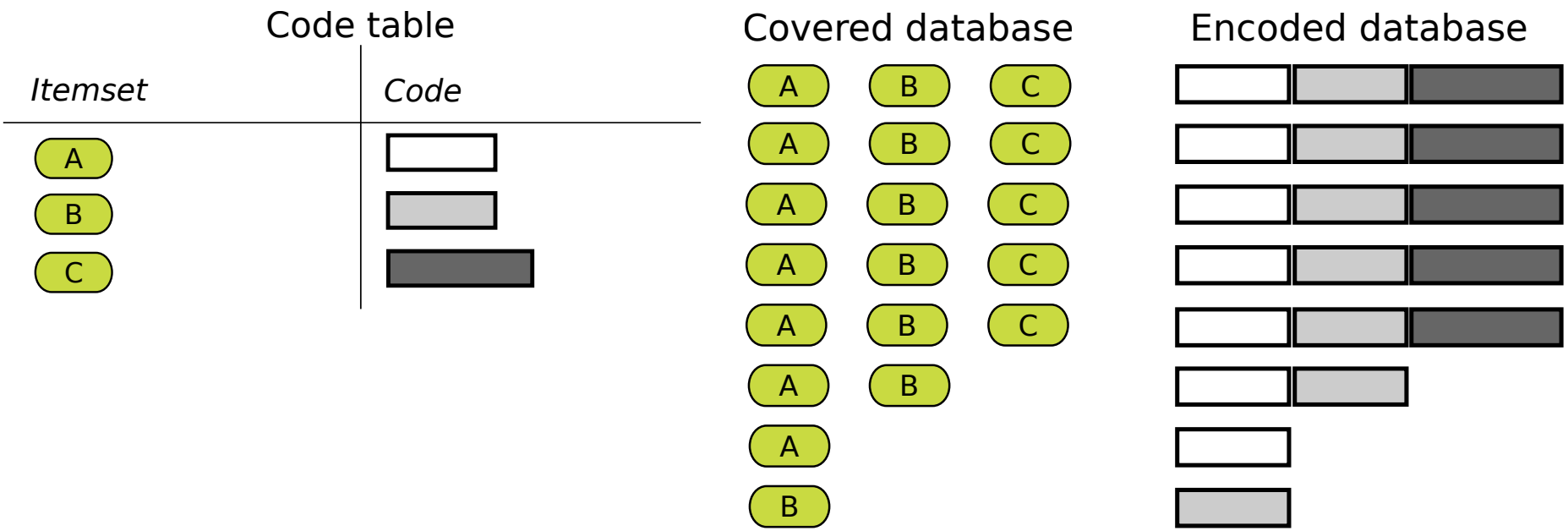
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table



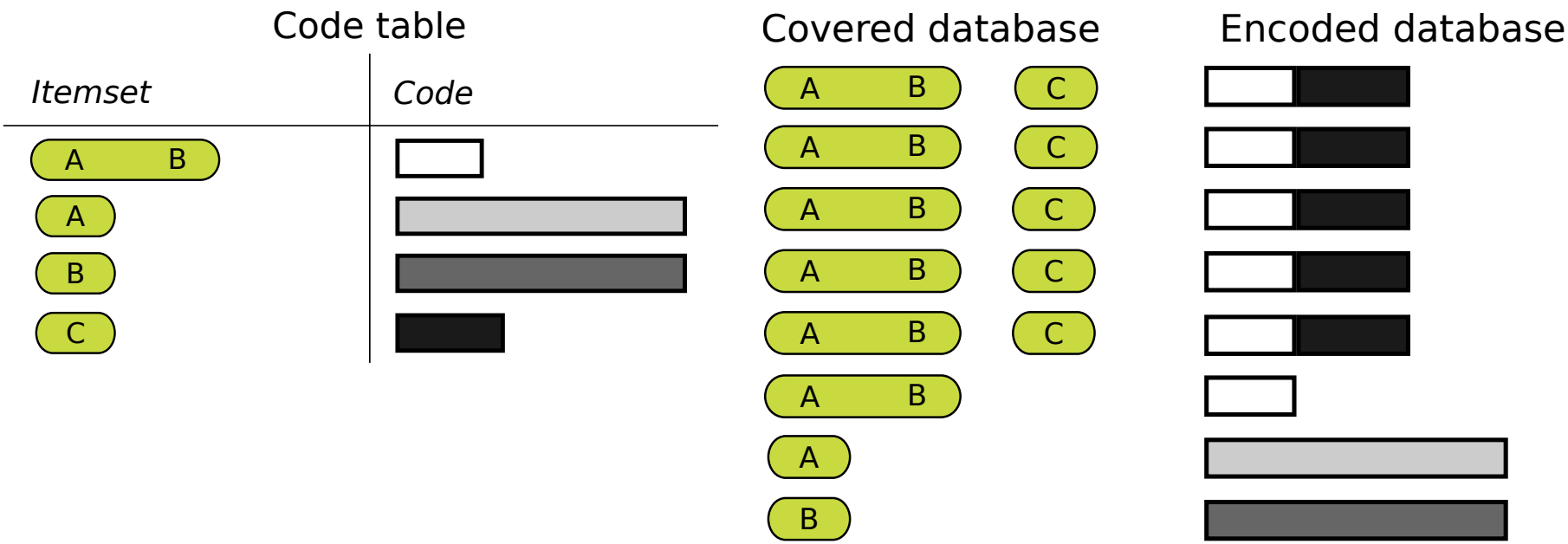
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression



Greedily construct code table bottom-up

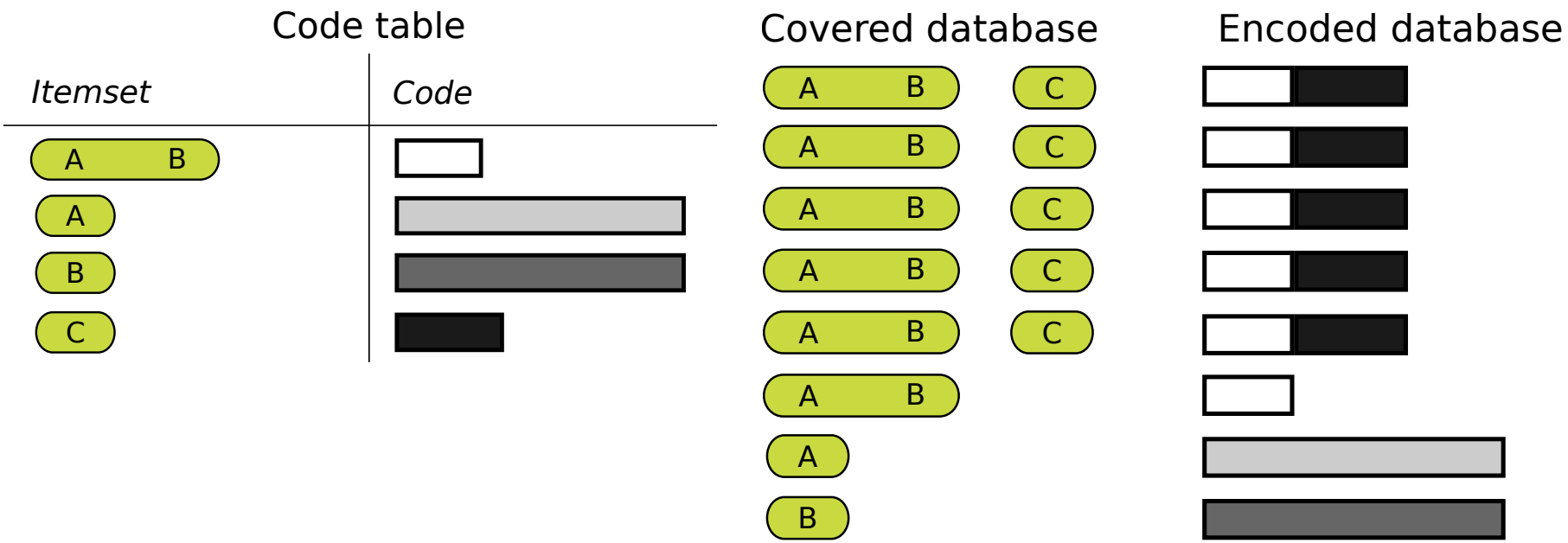
- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression



Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression

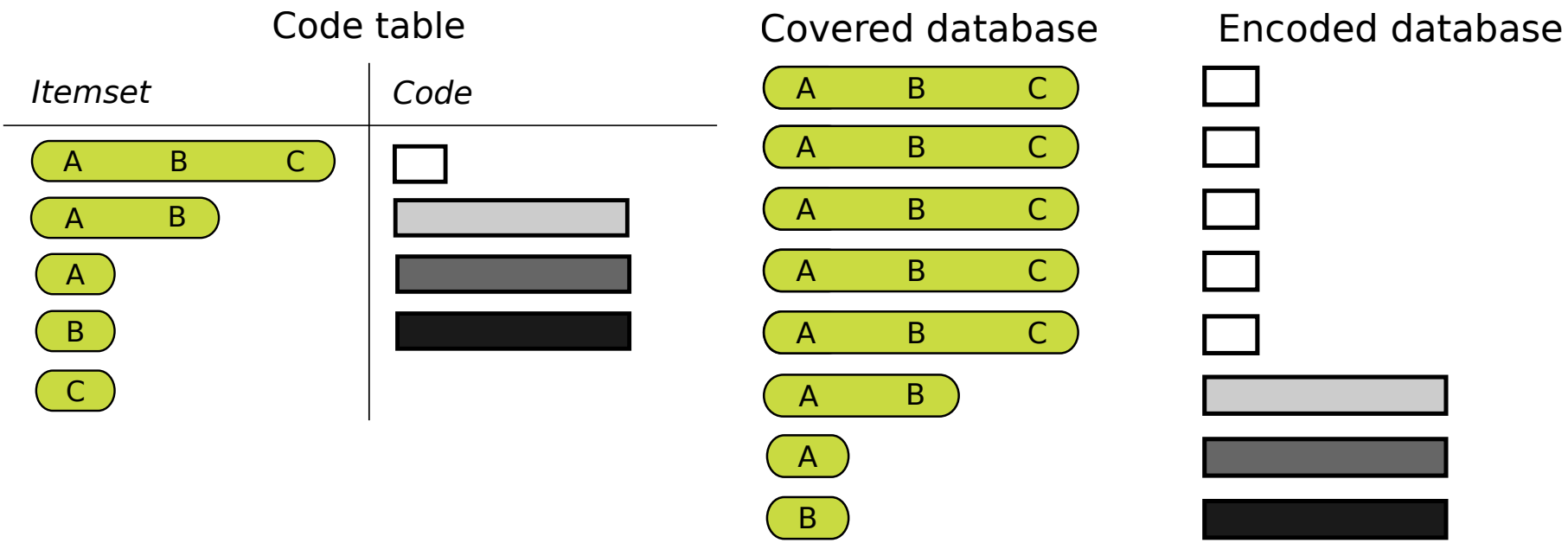
- Continue to refine current code table



Greedily construct code table bottom-up

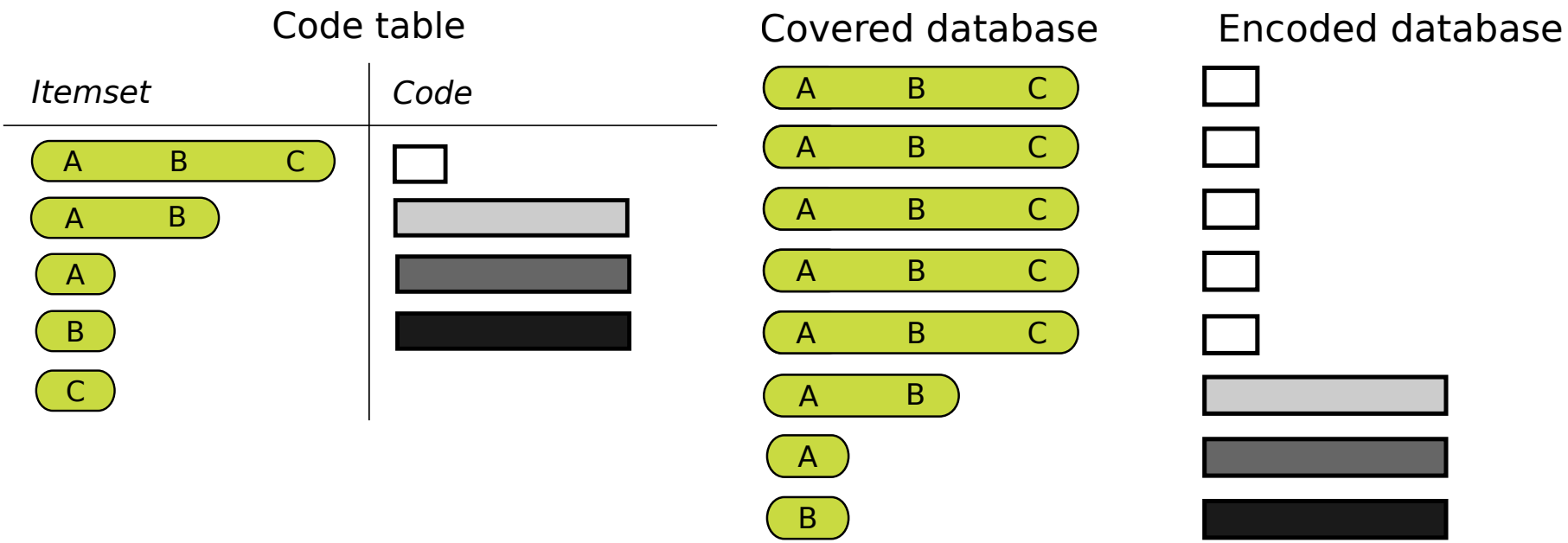
- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression

- Continue to refine current code table



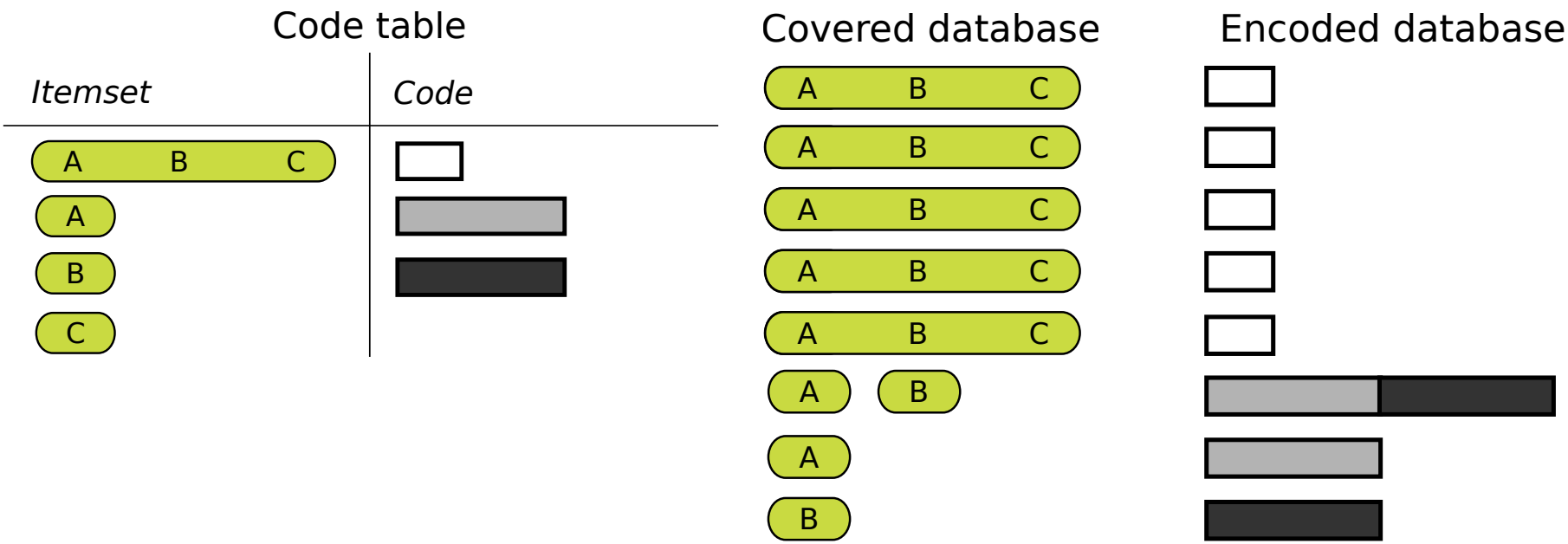
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression
  - Remove code table elements that no longer contribute
- Continue to refine current code table



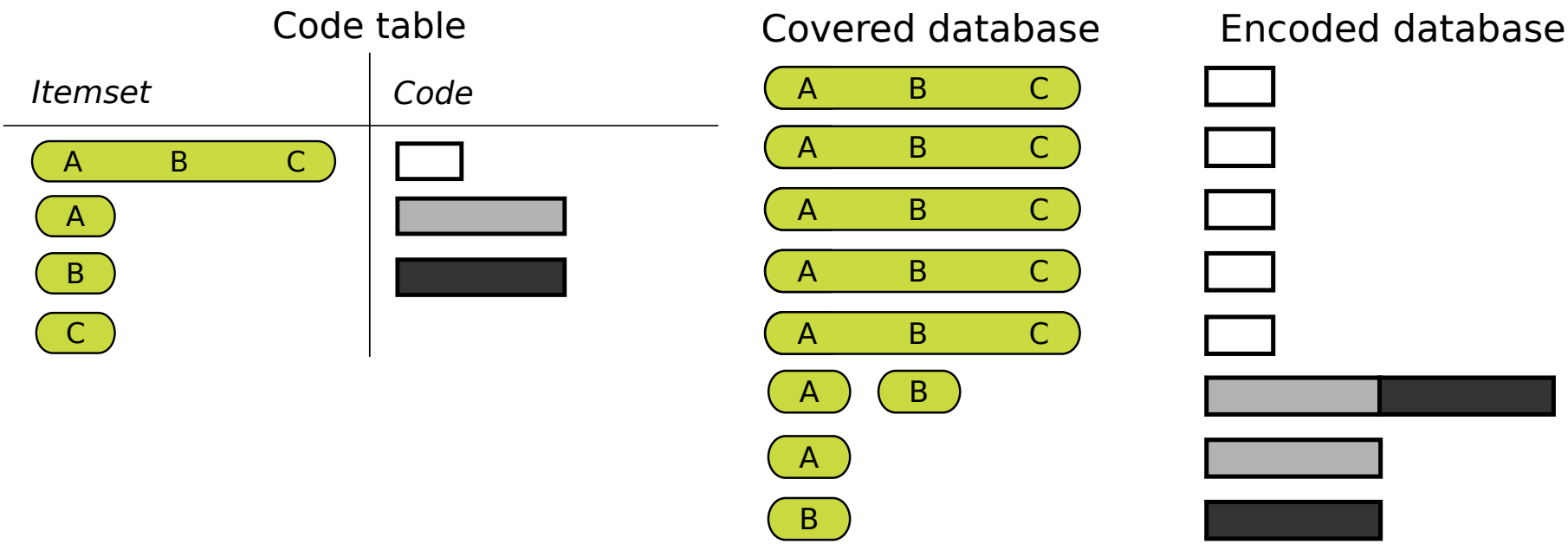
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression
  - Remove code table elements that no longer contribute
- Continue to refine current code table



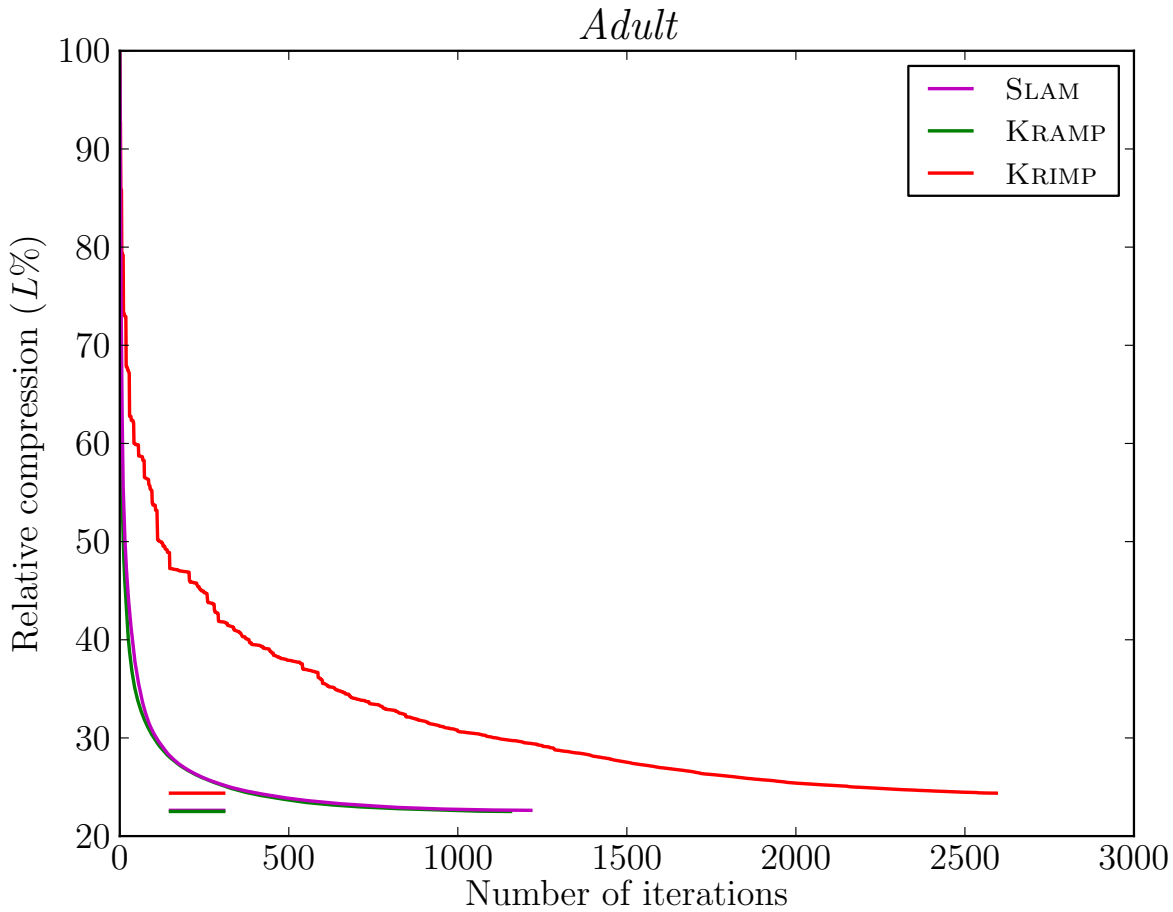
Greedily construct code table bottom-up

- Start with code table containing only singletons
- Optimise current code table locally
  - Consider all pairwise combinations of itemsets in code table
  - Add candidate with highest gain in total compression
  - Remove code table elements that no longer contribute
- Continue to refine current code table, or stop



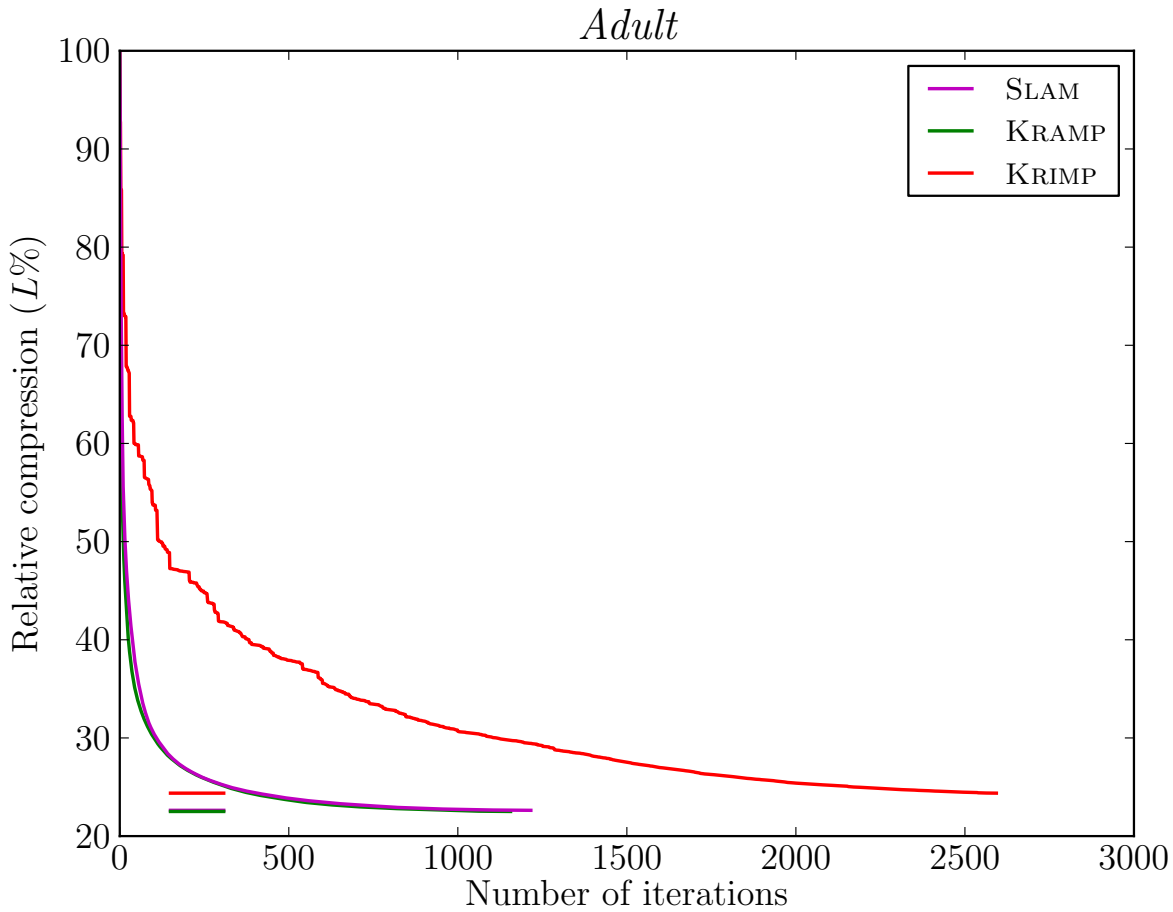
Greedily construct code table bottom-up

- Selecting the candidate pair with highest gain
- Converging quickly to better compression



Greedily construct code table bottom-up

- Selecting the candidate pair with highest gain is expensive
  - Need to compress database for each candidate
- Converging ~~quickly~~ to better compression takes 1 week

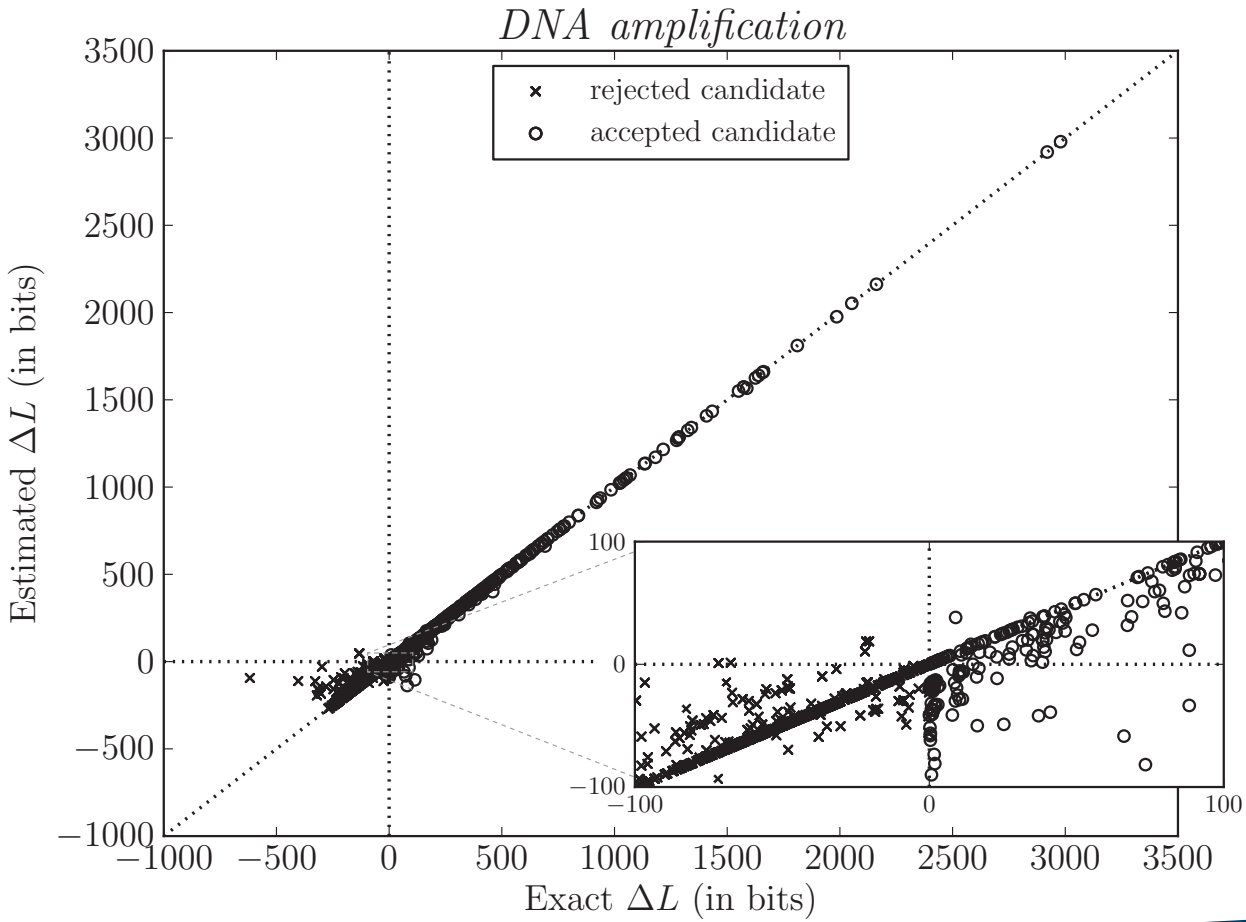


Accurate and efficient heuristic to estimate gain

- Calculate gain through usage counts of code pairs
  - Disregard effects on usage of other codes

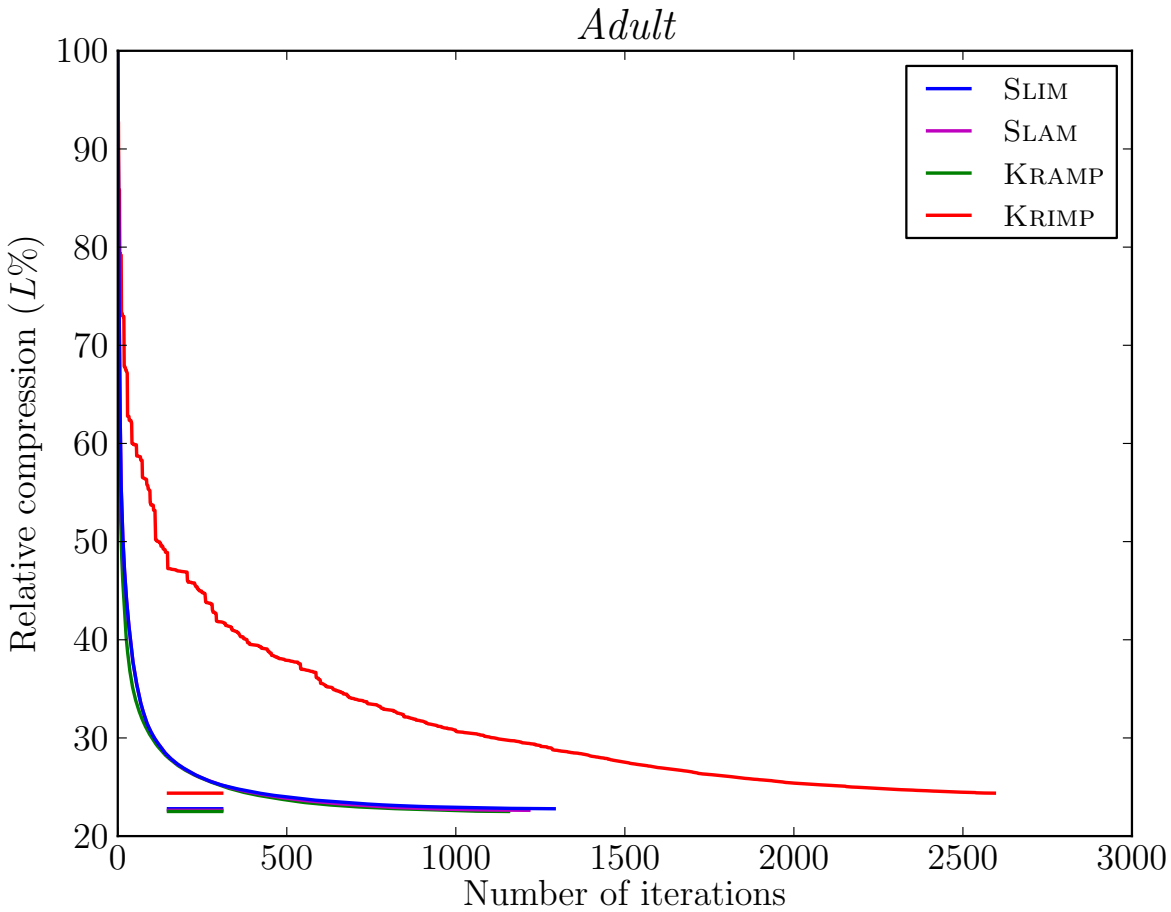
Accurate and efficient heuristic to estimate gain

- Calculate gain through usage counts of code pairs
  - Disregard effects on usage of other codes



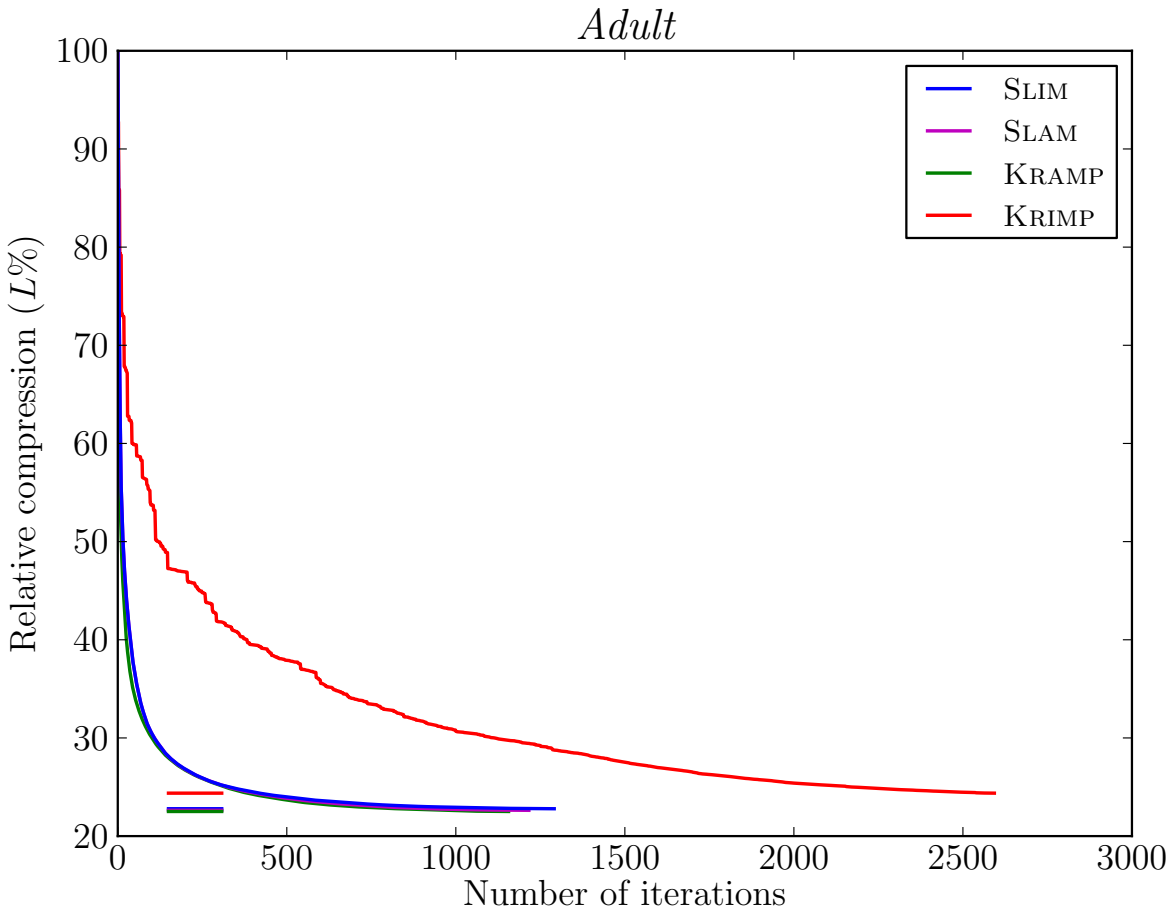
Accurate and efficient heuristic to estimate gain

- Calculate gain through usage counts of code pairs
  - Disregard effects on usage of other codes



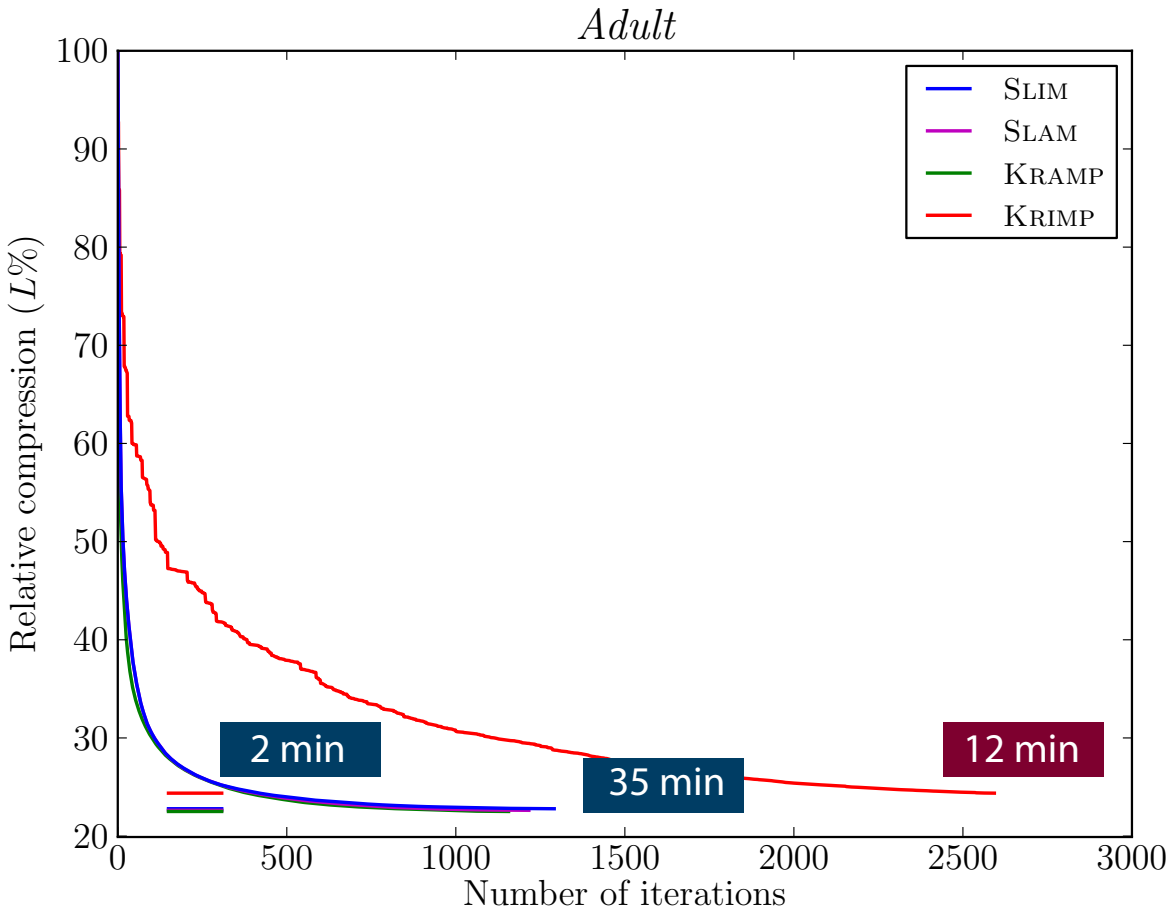
Accurate and efficient heuristic to estimate gain

- Calculate gain through usage counts of code pairs
  - Disregard effects on usage of other codes
- Use branch-and-bound to find highest estimated gain



Accurate and efficient heuristic to estimate gain

- Calculate gain through usage counts of code pairs
  - Disregard effects on usage of other codes
- Use branch-and-bound to find highest estimated gain

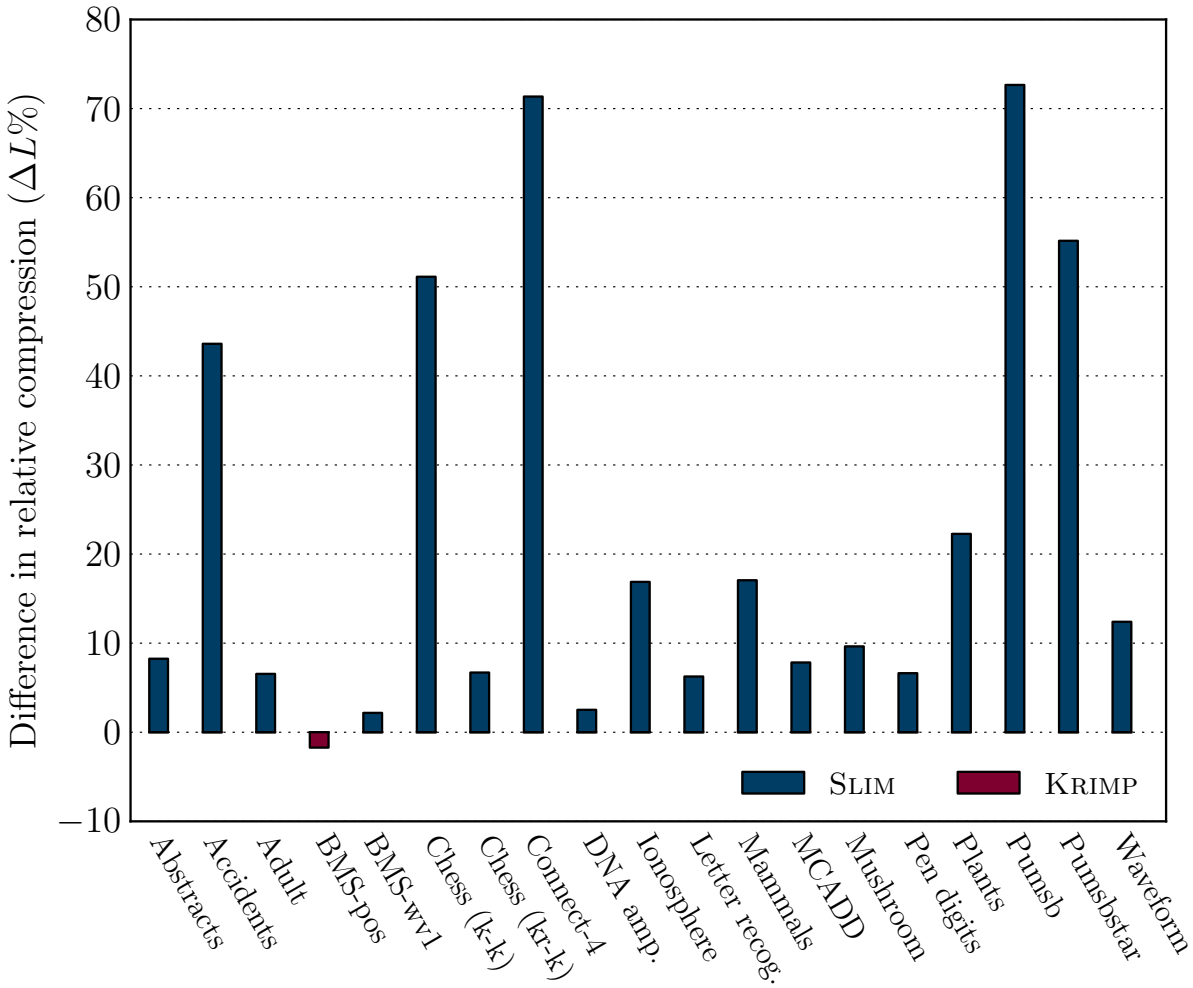


SLIM vs. KRIMP: setup

- Use wide range of benchmark and real datasets
- Limit processing time to 24 hours
- For KRIMP, mine itemsets with lowest feasible minsup

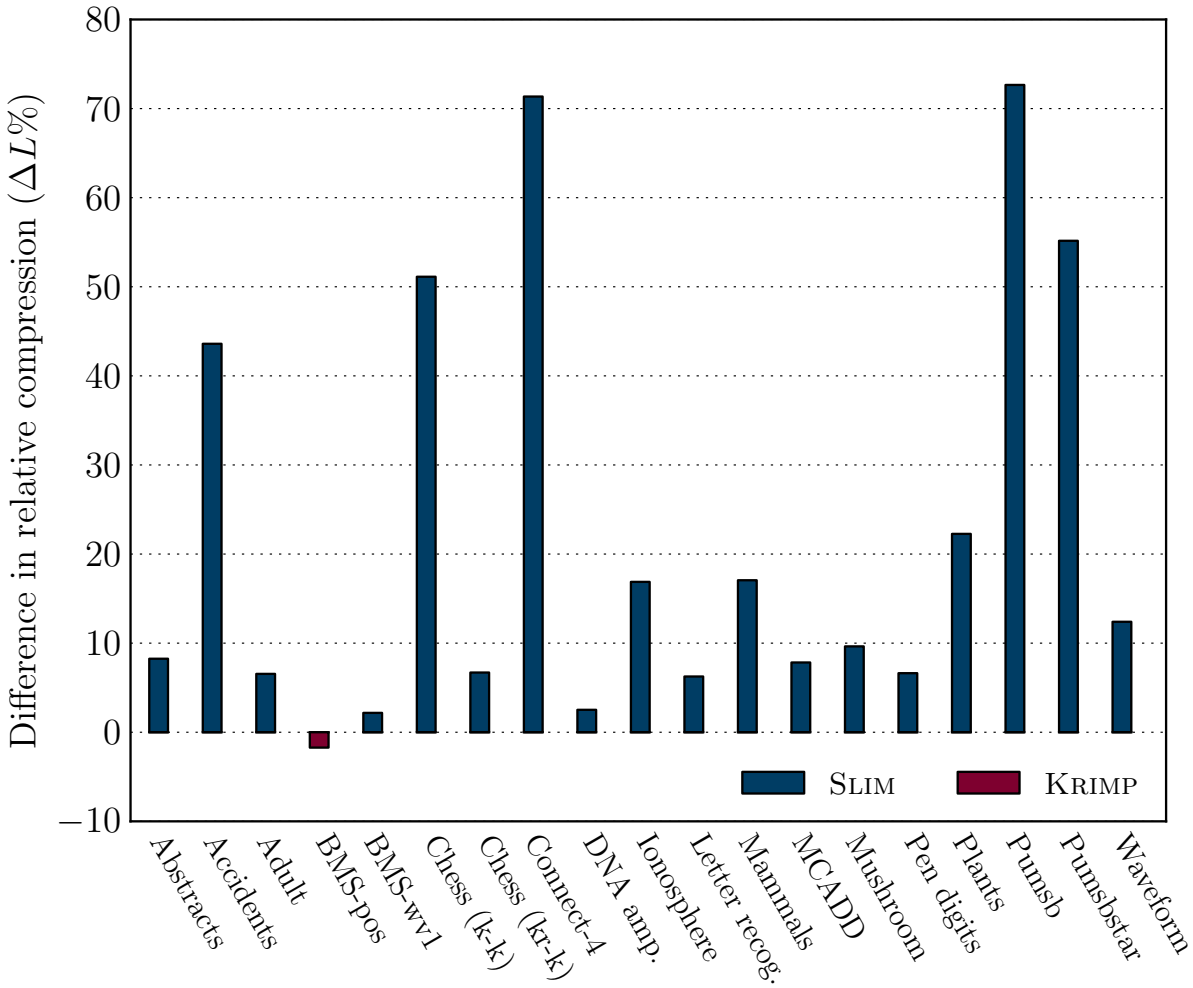
Better compression

comparing results after at most computing 1 day



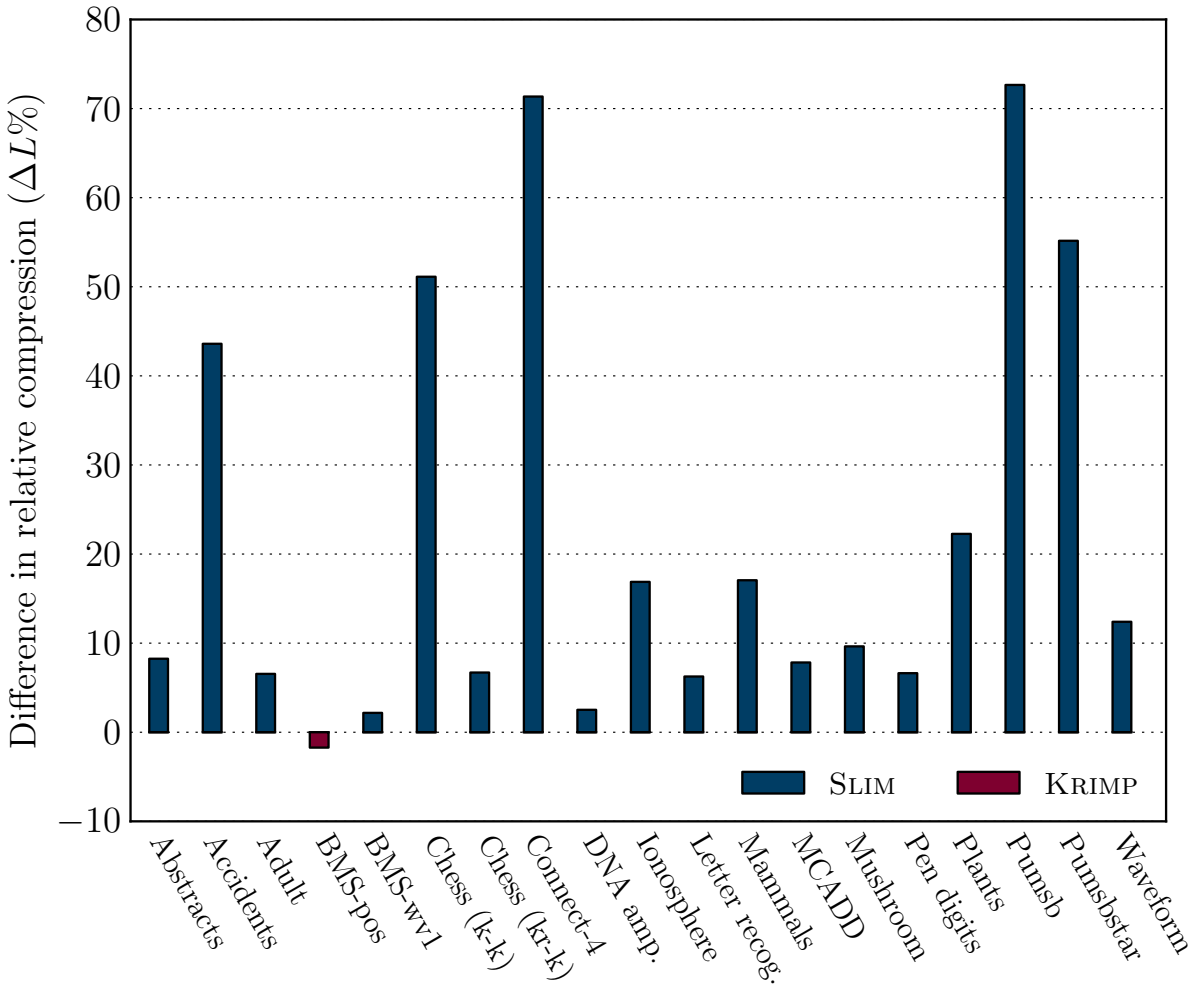
Better compression
comparing results after at most computing 1 day

- High difference



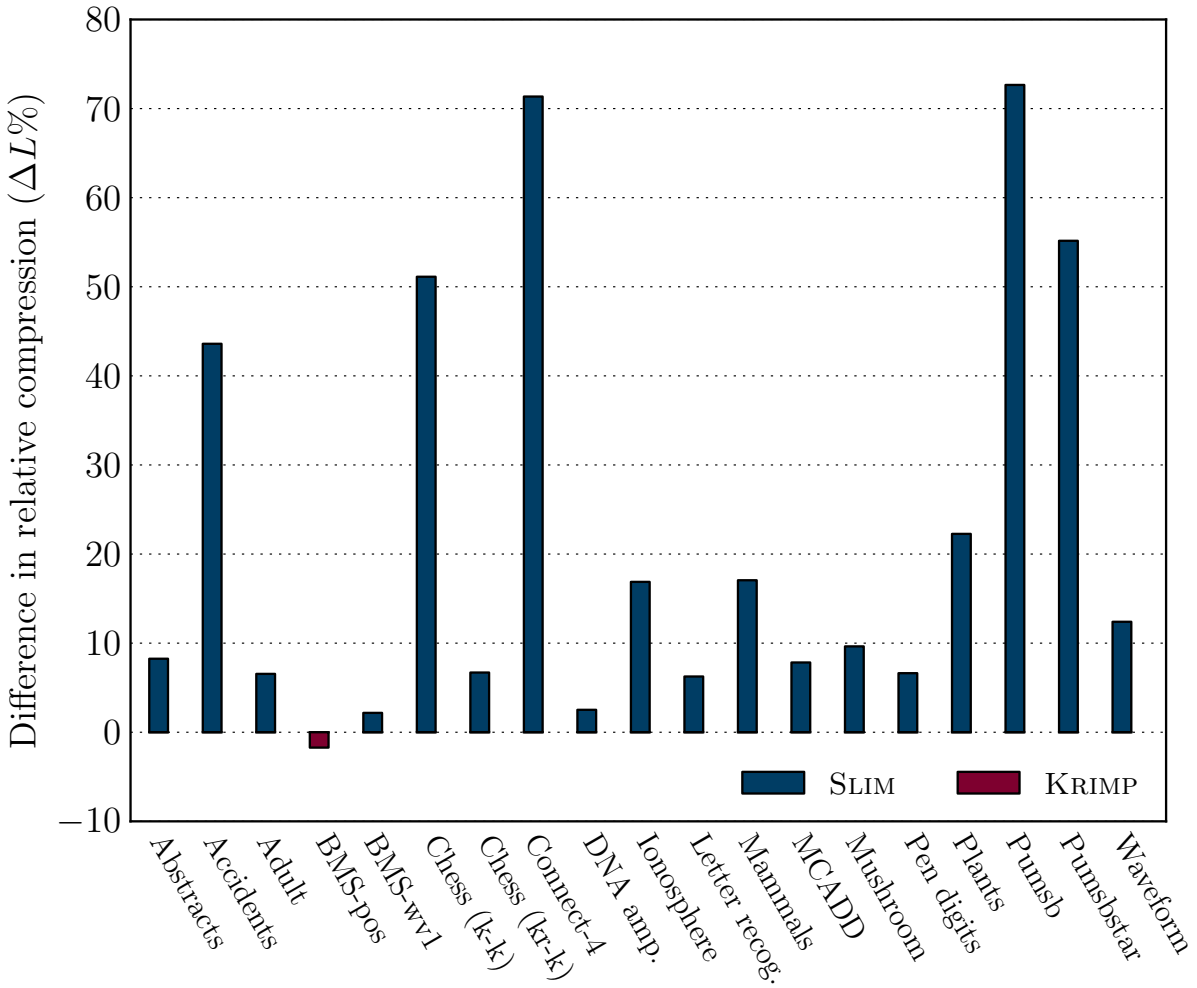
Better compression
comparing results after at most computing 1 day

- High difference $\rightarrow$ mine at lower minsup threshold



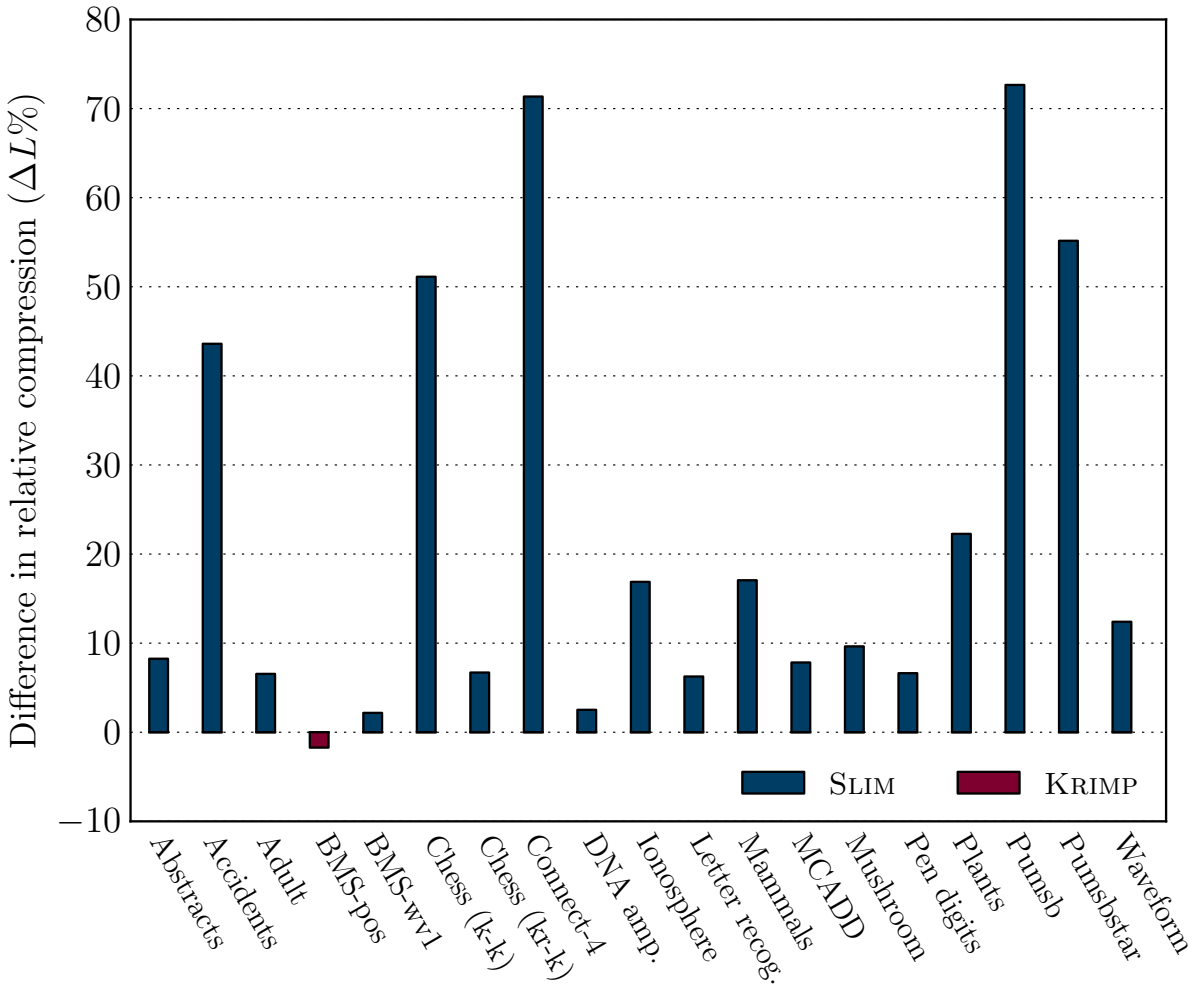
Better compression
comparing results after at most computing 1 day

- High difference $\rightarrow$ mine at lower minsup threshold
  - Impossible to mine all of those



Better compression
comparing results after at most computing 1 day

- High difference → mine at lower minsup threshold
  - Impossible to mine all of those, need only a few good ones



Code tables at least as characteristic
validated using classification experiment

Code tables at least as characteristic
validated using classification experiment

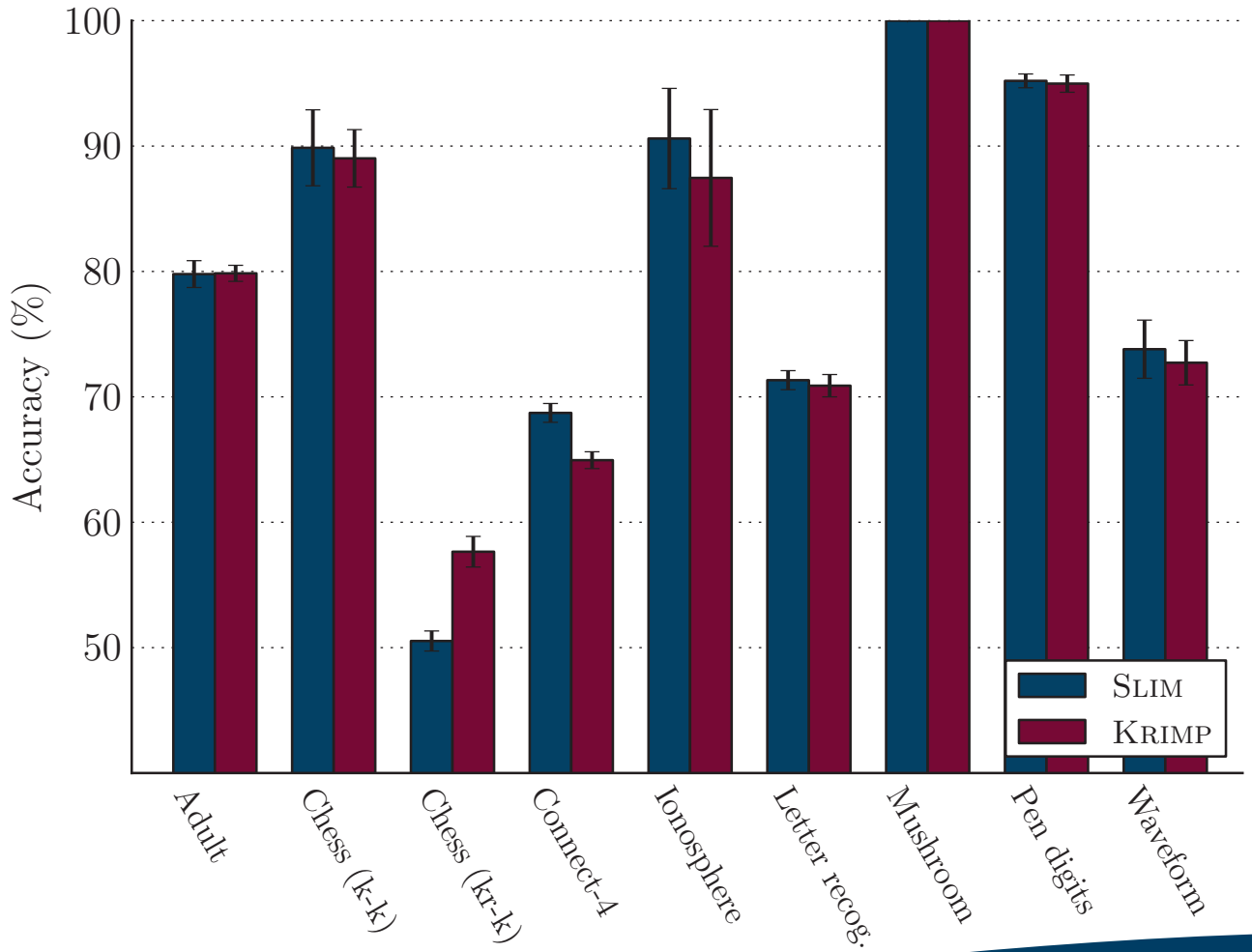
- Split data and build code table per class

Code tables at least as characteristic
validated using classification experiment

- Split data and build code table per class
- Assign class label based on encoded length

Code tables at least as characteristic
validated using classification experiment

- Split data and build code table per class
- Assign class label based on encoded length



SLIM vs. KRIMP: highlights

- Describing data more succinct

SLIM vs. KRIMP: highlights

- Describing data more succinct
- Providing high-quality descriptions

SLIM vs. KRIMP: highlights

- Describing data more succinct
- Providing high-quality descriptions
- Instantiating fewer candidate patterns

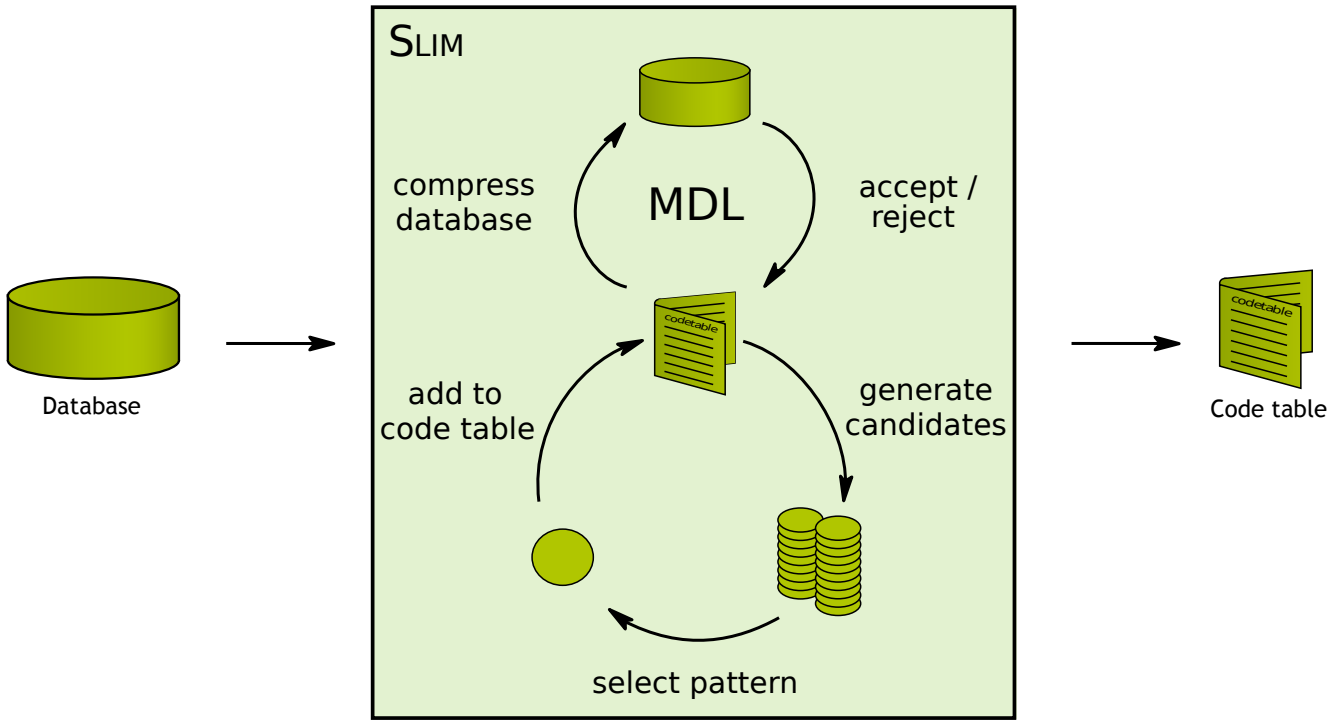
SLIM vs. KRIMP: highlights

- Describing data more succinct
- Providing high-quality descriptions
- Instantiating fewer candidate patterns
- Converging faster

SLIM vs. KRIMP: highlights

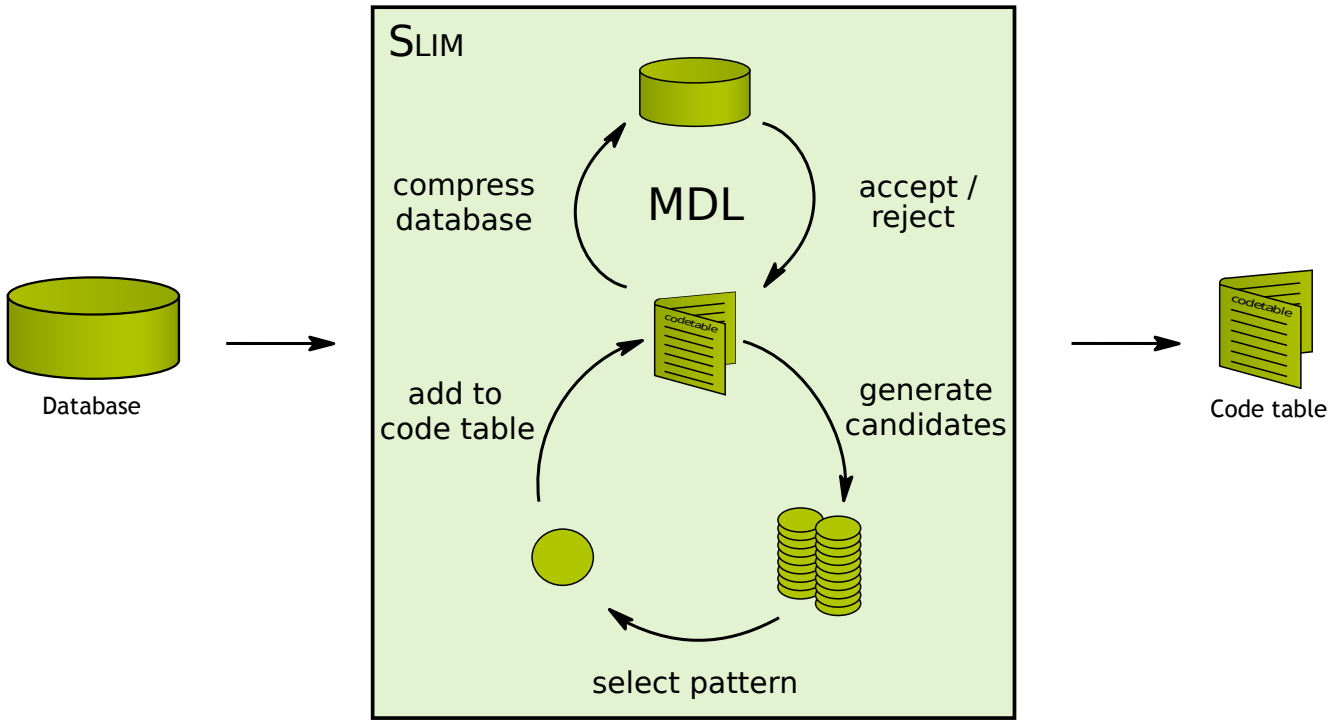
- Describing data more succinct
- Providing high-quality descriptions
- Instantiating fewer candidate patterns
- Converging faster
  - most time is invested in tail of convergence

SLIM: Directly Mining Descriptive Patterns



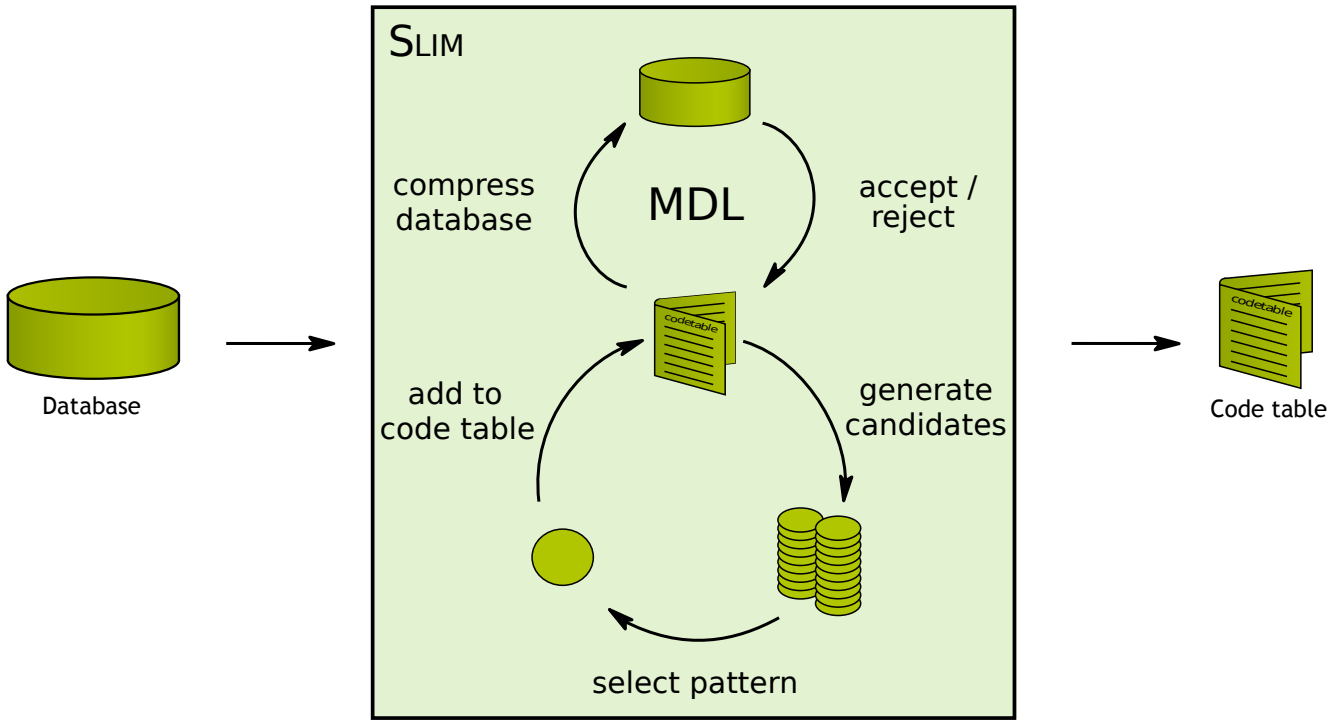
- Iteratively refining current description of the data
- Reconsidering candidates providing highest estimated gain

SLIM: Directly Mining Descriptive Patterns



- Iteratively refining current description of the data
- Reconsidering candidates providing highest estimated gain
- Any-time & parameter-free

SLIM: Directly Mining Descriptive Patterns



- Iteratively refining current description of the data
- Reconsidering candidates providing highest estimated gain
- Any-time & parameter-free
  - Providing detail only when necessary
  - Feasible to analyse large and dense data in more detail

For implementation and further reading

K. Smets & J. Vreeken.
SLIM: Directly Mining Descriptive Patterns.
Proceedings of the SIAM International Conference on Data Mining (SDM), SIAM, 2012.
Implementation available at `http://adrem.ua.ac.be/slim`.

J. Vreeken, M. van Leeuwen & A. Siebes.
KRIMP: Mining Itemsets that Compress.
Data Mining and Knowledge Discovery, 23(1):169–214, Springer, 2011.