Generic Mining of Condensed Pattern Representations under Constraints

Sergey Paramonov[1] Tao Chen[1] Tias Guns[2]

[1] KU Leuven, Belgium
[2] Vrije Universiteit Brussel, Belgium
sergey.paramonov@kuleuven.be, tao.chen1@student.kuleuven.be, tias.guns@vub.ac.be

Abstract

Our goal is to design and develop a principled approach to generic constraint-based pattern mining of any pattern type. While implementation details for different pattern types are widely different, the principles with respect to enforcing constraints are similar. We focus specifically on local constraints (size, cost, structure) and condensed representations (closed, free, maximal), whose combination is not straightforward. Our approach is inspired by inductive databases and combines ideas from two research directions in pattern mining: the development of specialized algorithms and the use of generic databases or constraint processing methods. It builds on top of existing highly efficient systems tailored for only one particular task and generalizes it to a wider class of problems using filtering techniques. As a result we obtain the *hyppro* system (from Hybrid Pattern Processing system; pronounced [hʌiprɔː]) which can efficiently mine patterns under both local and condensed representation constraints, for the three most popular datatypes: graphs, sequences and itemsets. The resulting system can outperform generic pattern mining search methods through novel filtering techniques for condensed representations.

1 Introduction

Pattern mining is a subfield of data mining concerned with finding patterns, regularities, in large datasets. There has seen tremendous progress in the development of highly scalable algorithms, starting with the Apriori algorithm for mining frequent itemsets [AIS93] and on to algorithms for sequences, trees, graphs etc with applications in market basket analysis, event log mining, drug discovery and bioinformatics [AH14].

A common drawback of pattern mining methods is that, although scalable, they typically produce too many patterns. This has lead to the development of constraint-based pattern mining methods [NZ14]. The idea is to take into account domain expertise and user-specific constraints to reduce the number of patterns. We can identify two major categories of constraints: 1) *local constraints*, these are constraints on individual patterns such as regarding their size or length, their cost in case costs are associated to pattern elements, or other structural constraints; and 2) *condensed representation constraints*, these are constraints aimed at reducing the redundancy in the overall pattern set, for example by disregarding patterns that are sub/superpatterns of another and have the same frequency (closed/free patterns), or by keeping only the maximal frequent patterns according to the superpattern relation.

Within constraint-based mining, there are two streams of work: methods that make specialised methods more generic by adding support for specific constraints to the algorithms [PHM00; YHA; YH03]; and methods that use generic search techniques which are often specialised to be more efficient for mining large datasets [RGN08; NDGN13; AGS16; GGQRS].

In: S. Hölldobler, A. Malikov, C. Wernhard (eds.): *YSIP2 – Proceedings of the Second Young Scientist's International Workshop on Trends in Information Processing, Dombai, Russian Federation, May 16–20, 2017*, published at http://ceur-ws.org.

However, both approaches are typically pattern-type specific. Furthermore, these methods focus either on local constraints *or* condensed representations, e.g. the approaches based on set systems (with polynomial-delay) [BHPW10; AU] that are applicable to multiple datatypes do not allow reasoning under local constraints and focus only on a particular type of condensed patterns such as closed. Combining the two requires more care as the order in which the constraints are enforced matters [BL06].

What is missing is a unified approach that works for both local constraints and condensed representations, and with any pattern type. Indeed, the principles for different pattern types are similar and would merit from a generic methodology, even if the low-level implementations are different.

Our approach borrows from the inductive databases concept [IM96] in which data and patterns are treated as equal first-class citizens, and where both mining and querying/filtering are considered basic operations. A number of inductive query mining and database techniques have been proposed [IV99; BCFGPR; RJLM02], which focused on integration with SQL and existing database technology. We do not limit ourselves to the relational algebra or table representations, but maintain the vision that pattern mining is the process of applying a sequence of operators to data/patterns alike. Following this vision we propose a 3-step framework consisting of mining all frequent patterns, enforcing local constraints followed by extracting a condensed representation. Each of the constraints can consist of multiple filtering steps as well as the use of indexes and other techniques inspired by databases and constraint processing.

The contributions of this paper are as follows:

- we propose a generic pattern-type agnostic framework for constraint-based pattern mining;
- we present a principled way of combining local constraints and condensed representation constraints;
- this includes a novel and highly effective multi-stage post-processing method for condensed representations.

We have implemented the proposed system for the pattern types of itemsets, sequence and graphs, each time using the best known frequent pattern miner of that type. In terms of generality, the system is on par with constraint-based mining systems developed for one pattern type, see Table 1. Furthermore, the experiments show that in many cases the additional time spent post-processing is reasonably small, and that our system can outperform generic search-based pattern mining systems that support both local constraints and condensed representations.

2 Preliminaries

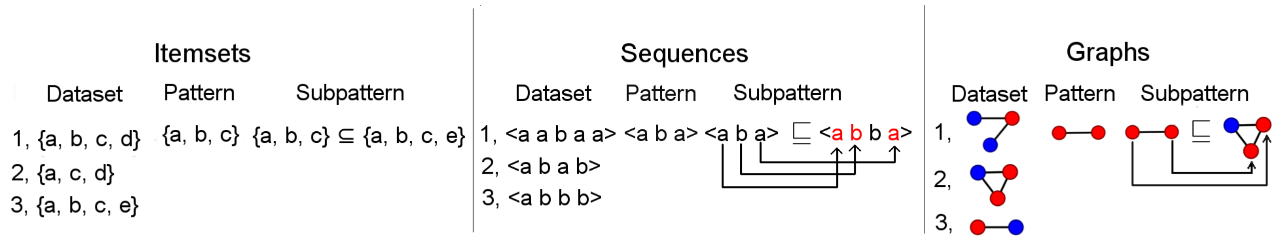


Figure 1: Visualization of pattern datatypes, datasets and corresponding pattern matching

In the foundational paper of Mannila and Toivonen [MT97] pattern mining is presented as the task of finding $Th(\mathcal{L}, \mathbf{r}, q)$, given a language of patterns $\mathcal{L}$, a transactional database $\mathbf{r}$ and a selection predicate q, such that $Th(\mathcal{L}, \mathbf{r}, q) = \{\varphi \in \mathcal{L} \mid q(\mathbf{r}, \varphi) \text{ is true}\}$.

The language of patterns $\mathcal{L}$ determines the pattern *type*. This can be itemsets (sets of elements), sequences, graphs and so forth. The database $\mathbf{r}$ consists of multiple entries, often called *transactions* in pattern mining from the original application domain in customer purchase transaction analysis. A transaction is a tuple $(tid, datapoint)$ where *tid* is the identifier of the transaction and *datapoint* is the data associated with this transaction; it is typically of the same language/type as $\mathcal{L}$. The goal then is to find all patterns φ that appear in the data and satisfy constraint $q(\mathbf{r}, \varphi)$.

The most basic setting is that of frequent pattern mining, in this case, the constraint q states that each pattern φ must occur more than θ times in the data, where θ is some user-chosen threshold. More formally $q(\mathbf{r}, \varphi) = |\{t \in \mathbf{r} | \varphi \sqsubseteq t\}| \geq \theta$; $\sqsubseteq$ is the occurence or subpattern relation. It determines when a pattern is a subpattern of another pattern/*datapoint*.

We first look into more detail into the subpattern relation for different pattern types, after which we discuss local constraints and condensed representations.

Pattern types and subpattern relations

The definition of when a pattern is a subpattern of another pattern depends on the language $\mathcal{L}$ and hence the pattern type. We consider three pattern types in this work: itemsets, sequences and graphs. Figure 1 demonstrates examples of these three pattern types, including example datasets and a visualization of the subpattern relation.

In itemset mining, the subpattern relation is defined as follows: let an *itemset* I be a set of items $I \subseteq \{1, \dots, n\}$ and a dataset D be a set of pairs (tid, J), where *tid* is an identifier and J is an itemset. Then, we say that an itemset I occurs in transaction (tid, J) iff $I \subseteq J$.

In sequence mining, an ordered list of characters $\langle s_1, \dots, s_m\rangle$ is called a sequence. A sequence S is called a subsequence of another sequence C if all characters in S also appear in C and in the same order, with possibly many other characters in between two matching characters. More formally for $S = \langle s_1, \dots, s_m\rangle$ and $C = \langle c_1, \dots, c_n\rangle$: $S \sqsubseteq C$ iff $m \leq n$ and for all $i \in [1, m]$ there exist integers j_i s.t. $1 \leq j_1 \leq \cdots \leq j_m \leq n$ such that $c_{j_i} = s_i$. Hence, a sequence S occurs in transaction (tid, C) of dataset D iff $S \sqsubseteq C$.

In graph mining, a graph G is a triple (V, E, l) where is V is a set of vertices, E is a set of edges and l is a labeling function that maps each edge and each vertex to a label. We say that G is a subgraph of a graph $H = (V_h, E_h, l_h)$ iff G is isomorphic to a subgraph of H i.e. there exists an injective function f such that

- if $e \in E$, then $f(e) \in E_h$ and $l(e) = l_h(f(e))$
- if $v \in V$, then $f(v) \in V_h$ and $l(v) = l_h(f(v))$

We say a graph G occurs in a transaction (tid, H) of a dataset D iff G is isomorphic to a subgraph of H: $G \sqsubseteq H$.

While checking the subset relation is relatively simple, especially when itemsets are implemented as bitvectors, for graphs this requires solving the subgraph isomorphism problem which is NP complete [Coo71].

Local constraints

are constraints that each individual pattern must satisfy. For itemsets, this can be constraints on the length of the set, membership of an item or on the cost of an itemset in case every item has a known cost. For sequences, there can additionally be constraints expressed as regular expressions that the target sequences must satisfy and on the arities of the symbols. For graphs, an extra category involves constraints on the in/out degrees of the edges and the presence of particular subgraphs.

Obviously, these constraints can be combined using Boolean logic to obtain more complex constraints including conjunctions, disjunctions and conditional constraints. However, it remains that all local constraints can be verified on each pattern individually (hence *local*).

Condensed representations

The key idea is to remove redundancy by suppressing patterns with similar properties that are sub- or superpatterns of the other. The best known example are closed frequent patterns; a pattern is *closed* [PHM00] if there is no superpattern with the same frequency. Similarly, *free* patterns [NDGN13] are those for which there is no subpattern with the same frequency. Patterns are called *maximal* [RGN08] if there is no superpattern that also satisfies the constraints.

Condensed representation constraints can be seen as *pairwise* constraints checking if one pattern *dominates* the other (e.g. if it is a subpattern with the same frequency) [NDGN13].

When there are additional constraints at play, one can choose to first mine a condensed representation and enforce the local constraints on the remaining patterns. While computationally beneficial, this can have unintended side effects [BL06]. Consider the case of a database with only a single closed itemset of length n. If we would add the local constraint that each pattern should be at most length $m < n$, then no pattern would remain. The more natural interpretation is that condensed representation constraints should only apply to those patterns that satisfy all local constraints; that is, a pattern that does not satisfy all local constraints can not dominate another pattern. This is the interpretation we use in this work.

3 Method

We design a three step procedure illustrated in Figure 2. This can be seen as an inductive databases pipeline, where we start from a transactional dataset and run a specialized frequent mining algorithm to obtain the set of all frequent patterns. This set of patterns is as much a first-class citizen as the dataset is. The set of patterns is then further and further processed to obtain smaller sets of patterns. While not obvious in this figure, step 3 consists of multiple operations in itself for efficiency reasons. It should be clear that more operations could be easily added to this framework, both on the dataset and on the pattern sets in between or at the end.

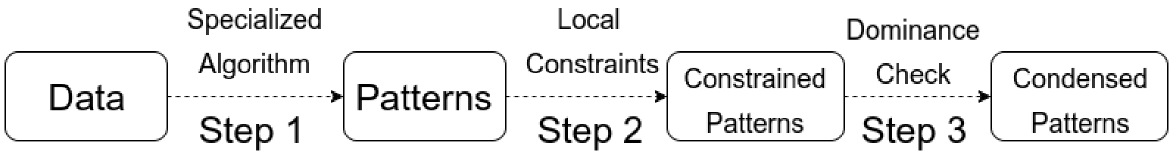


Figure 2: Three step constraint-based algorithm: Step 1 involves the optimized specialized algorithms, Step 2 enforces local constraints and Step 3 does the pairwise dominance checks to obtain the condensed representation.

3.1 The *hyppro* algorithm

Algorithm 1 shows the algorithm in more detail. We now explain the three major steps in more detail. Note that *patterns* is assumed to be an *ordered* list of patterns throughout the algorithm; *skip_set* is not.

Algorithm 1 *hyppro* algorithm for computing condensed patterns under constraints

```
 1: Input: A transactional database $D$ of type $t$, a frequency threshold $\theta$, the set of local constraints $C$ and a condensed representation constraint $r$
 2: Output: patterns – the set of condensed frequent patterns under constraints
 3: patterns ← run_miner$_t(D, \theta)$                                ▷ Step 1: frequent pattern mining
 4: patterns ← process_local_constraints(patterns, $C$)                  ▷ Step 2: local constraints
 5: ▷ Step 3: condensed representation constraint with dominance checks
 6: skip_set ← neighborhood_dominance_check(patterns, $r$)               ▷ 3a: Compare neighbor patterns
 7: for pattern in sort_by_length$_r$(patterns) do
 8:     if pattern ∈ skip_set then
 9:         continue
10:     candidates ← select_where$_r$(patterns \ skip_set)                ▷ 3b: Preserved properties
11:     for candidate ∈ candidates do
12:         if dominance_check$_r$(pattern, candidate) then               ▷ 3c: sub/superpattern check
13:             skip_set = skip_set ∪ {candidate}                          ▷ candidate is dominated by pattern
14: return patterns \ skip_set
```

3.1.1 Step 1: frequent pattern mining

In the first step, a highly efficient specialized algorithm is run to extract all frequent patterns from the data. Obviously, the system to run depends on the pattern type requested by the user.

Any mining frequent pattern mining system can be used which outputs both the frequent patterns and their frequency (or full cover set, but frequency is enough). Additionally, if the system is able to also output for each pattern what its direct subpattern is (as gSpan [YH02] does), then this information can be used in subsequent steps.

3.1.2 Step 2: local constraints

A property of local constraints is that all constraints can be checked on each pattern individually. Hence, in this step, we iterate over the set of frequent patterns once and check all local constraints on each pattern. Patterns that do not satisfy all constraints are removed. In practice, during this iteration we also build an index mapping each frequency value to the set of patterns with that frequency, for later use.

3.1.3 Step 3: condensed representation constraint with dominance checks

This is the most critical part. To illustrate, consider that the typical output of a graph pattern mining algorithm contains ten thousands graphs or more. This means that a naive dominance check would involve solving up to 100 million subgraph isomorphism problems as checking all patterns against all others requires $O(n^2)$ checks. This computational challenge creates a need for a better approach, knowing that many of these patterns will not be sub/superpatterns in any case.

The first key property that we use in our approach is that *pairwise dominance is transitive*: if a pattern dominates a subpattern, then that subpattern is not needed to discover the superpatterns that dominate the pattern, hence the subpattern can immediately be discarded.

We combine this with the fact that the highly efficient specialized miners do depth-first search, and they output the patterns the order they are found. This means that the patterns that are output before/after are more likely to be sub/superpatterns. Hence, in step 3a we first do a linear scan over all patterns and check whether they dominate any of the δ patterns that are output before or after this one. If so, those subpatterns are disregarded. Even with a reasonably small δ this already significantly reduces the set of patterns and hence the number of further checks that need to happen.

Note that in case a mining system outputs the pattern that is its immediate subpattern we can even avoid the dominance check and only (and much more cheaply) check whether it is an immediate sub/superpattern.

The second key property that we use is that a given *pattern can only dominate patterns that satisfy certain properties*, for example that a subpattern must be smaller in size. It is important to realize that a number of properties are preserved by the subpattern relationship for each datatype, meaning that if such a property does not hold for a given pattern and other pattern, then the pattern can never be a subpattern of the other pattern. For the *subset relation* there are three basic properties: a subpattern must be smaller in length and its minimal/maximal item must be greater/smaller or equal than the minimal/maximal one of the other itemset. For the *subsequence* relation, a subsequence has to be smaller than the other sequence, and each symbol in the subsequence must appear at least as many times in the other sequence as it does in the subsequence. For the *subgraph isomorphism* relation, we can represent the length of a graph as a pair *(#vertices,#edges)* and a subpattern must be Pareto-suboptimal, i.e. (x, y) is smaller than (x', y') iff $x < x'$ and $y \leq y'$ or $x \leq x'$ and $y < y'$; we also consider a similar property as for sequences, namely let (l_v, l_e, l_w) be a label-triplet with labels l_v, l_w of nodes v, w and their edge label l_e, then the arity of the label-triples must be at least as large in the other graph as in the subgraph.

We use this to order the set of patterns such that dominating patterns are more likely to be encountered first, thereby reducing the set of non-skipped patterns more quickly. More concretely, for closed/maximal it is ordered on decreasing size and for free patterns on increasing size. For graphs, one does not order on size alone but a partial order on $(\#nodes, \#edges)$ instead.

We also use it to efficiently select, in step 3b, from the set of all non-skipped pattern only those that can be dominated based on such easy-to-check properties as size and label/symbol arities. Notably, to check the closedness or free relation, patterns only have to be compared to patterns that have *the same frequency*. Following the inductive databases principles, we borrow a technique from the database community. Namely, we first index all patterns by frequency (in step 2); then, in step 3b we first query for all patterns with the same frequency and only iterate over those to verify the easy-to-check properties.

Steps 3a and 3b are visualized in Figure 3 for sequences and graphs, respectively:

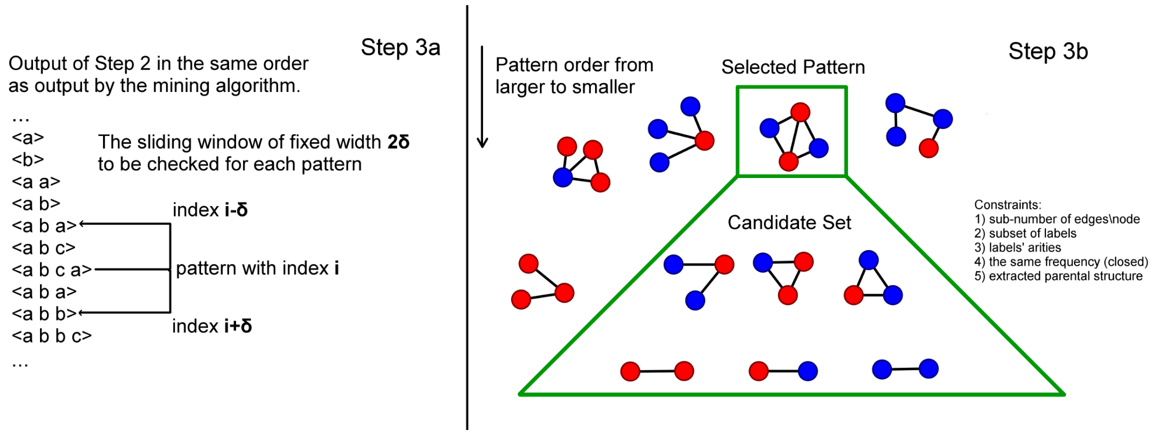

Step 3a
Output of Step 2 in the same order as output by the mining algorithm.
...
<a>
<b>
<a a>
<a b>
<a b a>
<a b c>
<a b c a>
<a b a>
<a b b>
<a b b c>
...
The sliding window of fixed width 2δ to be checked for each pattern
index i-δ
pattern with index i
index i+δ
Pattern order from larger to smaller
Selected Pattern
Step 3b
Candidate Set
Constraints:
1) sub-number of edges\node
2) subset of labels
3) labels' arities
4) the same frequency (closed)
5) extracted parental structure

Figure 3: Illustration of the dominance checking procedure; 3a: each pattern is first checked to the 2δ patterns around it; 3b: for a selected pattern (green box) the dominance properties are used to select only candidate dominated patterns (in the green triangle below) and only those are explicitly checked.

3.2 Extensibility

Adding a new local constraint for a pattern type amounts to writing a function that receives as input a pattern of that type and outputs yes or no. Adding a new condensed representation amounts to adding support for a new dominance relation on a pattern type. This hence requires implementing the dominance_check$_r$(*pattern*, *candidate*) function (which is also used in step 3a), as well as the select_where$_r$(*patterns* $\setminus$ *skip_set*) function.

To add a new pattern type, run_miner$_t(D, \theta)$ must be added to call a frequent pattern miner of that pattern type and parse its output appropriately. Then, local constraints and dominance relations must be added as appropriate.

4 Experimental Evaluation

We start by comparing *hyppro* to other state-of-the-art pattern mining systems in terms of constraints it supports. As Table 1 shows, our system is the only one that supports condensed representations and local constraints of any pattern type.

The key questions we next investigate in this experimental section are the following:

- **Q$_1$** What is the runtime distribution between three steps of the algorithm?

Table 1: Feature comparison between *hyppro*, specialized algorithms and dominance programming ("–" means "not designed for this datatype")

Datatype	**Task**	**gSpan**	**PPIC**	**ASP**	**Eclat**	**DP**	***hyppro***
Graph	frequent pattern mining	✓	–	–	–	–	✓
	condensed (closed, max, free)	✓	–	–	–	–	✓
	condensed under constraints	×	–	–	–	–	✓
Sequence	frequent pattern mining	–	✓	✓	–	–	✓
	condensed (closed, max, free)	–	×	✓	–	–	✓
	condensed under constraints	–	×	✓	–	–	✓
Itemset	frequent pattern mining	–	–	–	✓	✓	✓
	condensed (closed, max, free)	–	–	–	✓	✓	✓
	condensed under constraints	–	–	–	×	✓	✓

- **Q_2** How does the *hyppro* system improves performance over naive post-processing?
- **Q_3** How does the performance of *hyppro* compare to other generic pattern mining systems such as Dominance Programing and ASP sequence mining models?

All experiments have been run on a 64-bit Ubuntu machine with Intel Core i5-3570 CPU 3.40GHz and 8GB memory. The runtime limit is set to one hour. The code and all datasets will be made available as a github repository at

`http://github.com/SergeyParamonov.`

Firstly, in Question **Q_1**, we analyze the runtimes of the different steps. As specialized algorithms (step 1) we used state-of-the-art methods in pattern mining: Eclat, PPIC and gSpan*. Table 2 summarizes the results for multiple datasets for each datatype; we added a local dummy constraint that reads the pattern and always returns true. The runtime results for the 'free' condensed representation are similar to closed and not shown. As we see in most cases the post-processing is bound by a constant factor of 10-20 with respect to the specialized algorithm. Step 2 is typically very fast (subsecond), as it requires just a linear scan over all generated patterns. Step 3's expected runtime is harder to characterize as it depends on the effectiveness of filtering and the remaining number of dominance checks to do. In most cases in the table it takes seconds and for 5 cases it takes more than 10 seconds.

We next investigate Question **Q_2**, by measuring the effect of disabling different filtering steps in step 3. Disabling 3a corresponds to removing the sliding window filter on Line 6 of Algorithm 1; and disabling step 3b amounts to not using any properties, that is no sorting (Line 7) and no selection filtering (Line 10). As we see from Figure 4 performance is improved drastically over naive-postprocessing (both 3a and 3b disabled); for graph and sequence mining, pattern ordering and filtering (3b) give the major runtime improvement. While not visible on the plots, step 3a typically gives a 2-3 times speedup, especially for maximal patterns. Also for closed itemset mining step 3b plays a crucial role, especially the use of an index to fetch all patterns with the same frequency. For maximal itemset mining step 3b has little effect but 3a plays a crucial role here.

Thirdly, in Figure 5 we compared our system to existing generic search-based pattern mining systems: Dominance Programming (DP) for itemset mining using a constraint solver [NDGN13] and sequence mining with Answer Set Programming (ASP) [GGQRS]. As Figure 5c indicates, *hyppro* outperforms the ASP model on mining condensed representations (the runtime comparison has to be done on generated samples provided with the ASP solver [GGQRS]).

5 Discussion and Conclusion

While research on generic pattern mining techniques is still ongoing [PLDR15; NG; GDTNR13], these techniques always focus on one specific pattern type. Furthermore, local constraints and condensed representations are usually studied separately [PHM00; YHA; YH03]. A notable exception is in the use of constraint programming for itemset mining [NDGN13; RGN08], of Answer Set Programming for sequence mining [GGQRS], and in a theoretical study of operators in structured pattern mining [GPN16].

We proposed a pattern-type agnostic approach. Closest in spirit to this is the C++ template library for pattern mining [CHSZ08]. However, the goal there is to define pattern-type agnostic building blocks that are used in a generic depth first

*`http://www.borgelt.net/eclat.html`
`http://sites.uclouvain.be/cp4dm/spm/`
`https://github.com/Jokeren/DataMining-gSpan`

Type	Dataset	Step 1		Step 2	Step 3		
		time (s)	#patt.	time (s)	representation	time (s)	#patt.
graphs	Chemical 340	0.74	844	0.03	closed	1.21	632
					maximal	1.34	105
	Compound 422	4.87	14486	0.08	closed	9.43	1248
					maximal	14.82	277
	nctrer	4.02	16406	0.01	closed	37.82	2187
					maximal	89.99	627
	yoshida	0.57	11361	0.01	closed	1.39	1021
					maximal	4.06	298
	bloodbarr	1.11	11817	0.04	closed	3.41	1380
					maximal	6.11	308
sequences	IPRG neg	1.32	5182	0.01	closed	0.20	5182
					maximal	1.54	3804
	IPRG pos	1.21	4659	0.01	closed	0.13	4659
					maximal	1.33	3057
	JMLR	5.88	56067	0.01	closed	48.48	55636
					maximal	276.40	40461
	Unix users	4.38	10089	0.01	closed	0.67	10078
					maximal	1.88	2370
	FIFA	18.63	40642	0.03	closed	0.62	40642
					maximal	94.69	24624
itemsets	zoo	0.91	133165	0.11	closed	1.84	2910
					maximal	0.70	214
	vote	0.28	47126	0.02	closed	2.82	27120
					maximal	0.61	1949
	tic-tac-toe	0.04	336	0.01	closed	0.03	334
					maximal	0.01	175
	splice	0.17	2068	0.01	closed	0.08	2067
					maximal	0.09	1116
	soybean	0.24	37419	0.02	closed	0.91	4122
					maximal	0.19	420
	primary-tumor	0.62	71946	0.17	closed	7.11	44744
					maximal	1.34	2832
	mushrooms	1.86	298751	0.16	closed	2.77	4564
					maximal	1.66	485
	german-credit	1.77	1162250	0.14	closed	125.84	65994
					maximal	372.63	28461

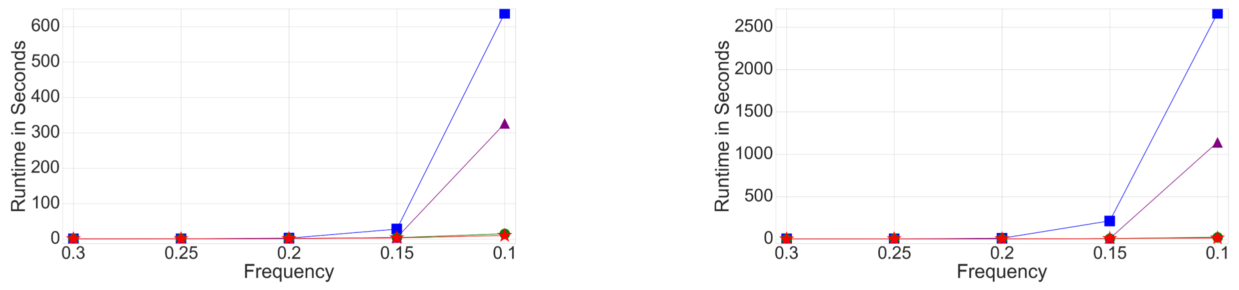
Table 2: Runtime distribution (in s) between the steps, for graphs and itemsets frequency is 0.1 (0.2 for german-credit), for sequences 0.02 (0.1 for FIFA)

search. In contrast, we chose to reuse highly optimized search methods and allow to extend the pipeline with local and condensed representation constraints. Indeed, a key asset of the pattern mining community is the development of highly optimized frequent mining algorithms [YH02; ZPOL97; AGS16]. A principled and generic way of reusing them can strengthen this position.

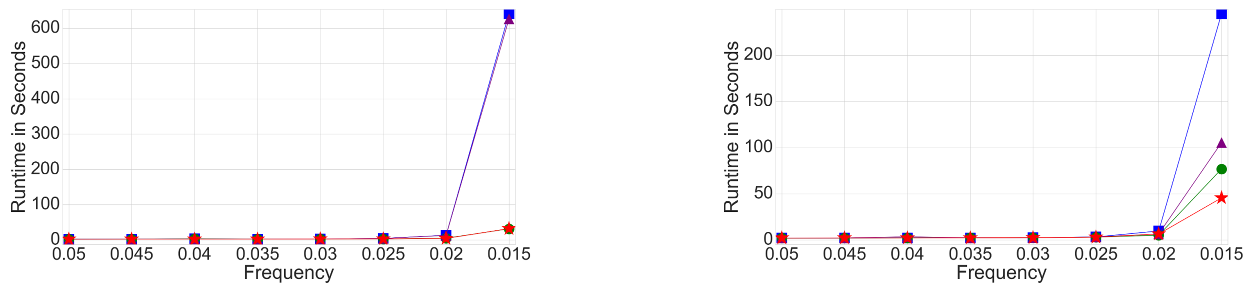
Another related system, from an inductive databases point of view, is the MiningZinc system [GDTNR13]. It provides a constraint-based modeling language and the system can automatically detect whether a specialized mining algorithm can be used to mine all patterns satisfying a subset of the constraints, while using generic constraint solvers to post-process the rest. Our method is less generic in using only a predefined (but extensible) set of building blocks, but it also supports condensed representations and is pattern-type agnostic. From a user perspective all the user has to do is provide a dataset and choose the frequency value, the local constraints and the condensed representation.

To summarize, we presented a pattern type agnostic framework for constraint-based pattern mining. Key to it is the inductive databases view, under which it can be seen as a pipeline of pattern filtering techniques. To efficiently support condensed representation constraints we use a number of novel techniques to avoid comparing all patterns to each other. Also, our method allows to combine local constraints with dominance checks, which few systems support.

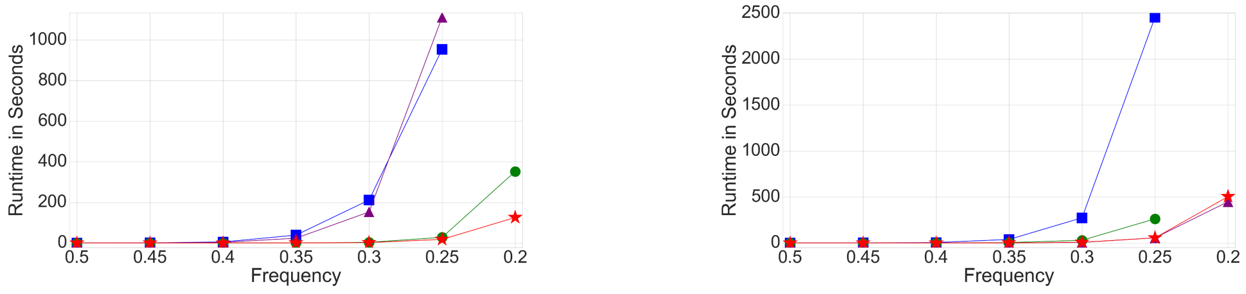
Another trend in pattern mining research is to develop methods that extract a very small set of patterns that *together*



(a) Graph Mining: AIDS Compound 422. Closed (left), Maximal (right)



(b) Sequence Mining. Unix Users. Closed (left), Maximal (right)



(c) Itemset Mining. German Credit. Closed (left), Maximal (right)

Figure 4: Comparison *hyppro* (red-star) with 3a disabled (green-circle), 3b disabled (purple-triangle) and naive postprocessing (blue-square): for graphs (top), sequences (middle), itemsets (bottom)

make a good pattern set [GGM04; VLS11; BKS10]. These can be seen as adding constraints on the entire collection of patterns. While this is outside the scope of this work, it should be noted that most of these methods actually post-process a set of frequent (closed) patterns, hence they could be added to our pipeline.

While parallelizing search algorithms is hard, we would like to point out that our approach could be parallelized quite easily and there is much room for engineering. Since each step essentially filters or transforms the pattern set, one could do parallel batch processing or use a framework like Apache Spark to handle this automatically.

This work has been partially funded by the ERC AdG SYNTH (Synthesising inductive data models) of prof. Luc De Raedt (KU Leuven).

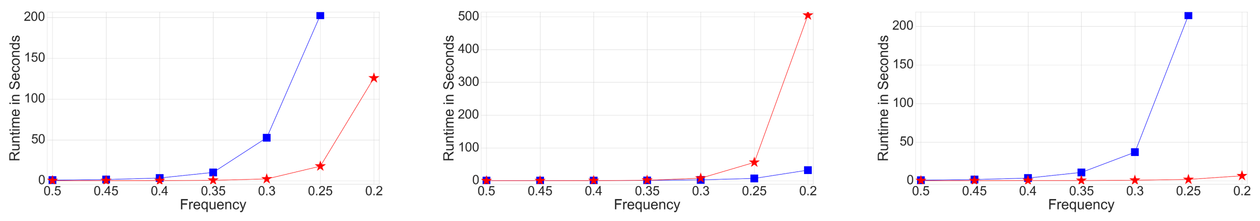
References

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *ACM SIGMOD*. ACM Press, 1993.

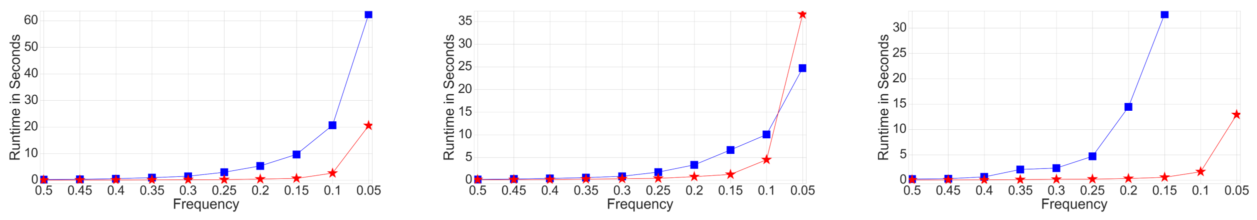
[AH14] Charu C. Aggarwal and Jiawei Han, editors. *Frequent pattern mining*. Springer, 2014. ISBN: 978-3-319-07820-5.

[NZ14] Siegfried Nijssen and Albrecht Zimmermann. Constraint-based pattern mining. In, *Frequent pattern mining*, pages 147–163, 2014.

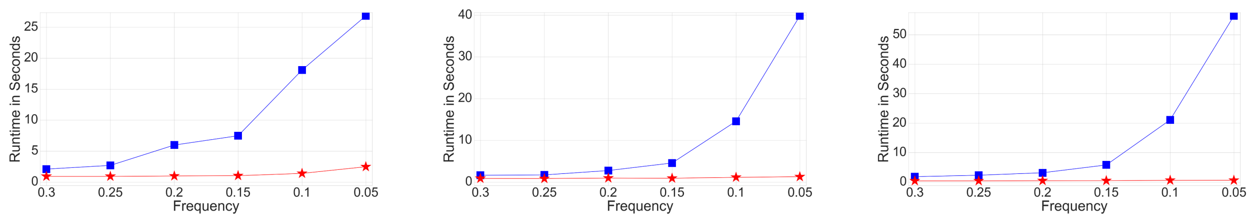
[PHM00] Jian Pei, Jiawei Han, and Runying Mao. CLOSET: an efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD workshop dmkd*, 2000, pages 21–30.



(a) DP (blue-square), *hyppro* (red-star). German Credit, Closed(left), Maximal(middle), Size+Closed(right)



(b) DP (blue-square), *hyppro* (red-star). Mushrooms, Closed(left), Maximal(middle), Size+Closed(right)



(c) ASP (blue-square), *hyppro* (red-star). Default generated sample for 100 (left), 200 (middle), 300 (right) sequences, Closed

Figure 5: Comparison with the other generic pattern mining systems: Dominance Programming (DP) [NDGN13] and ASP models [GGQRS]

[YHA] Xifeng Yan, Jiawei Han, and Ramin Afshar. Clospan: mining closed sequential patterns in large databases. In *SIAM 2003*, pages 166–177.

[YH03] Xifeng Yan and Jiawei Han. Closegraph: mining closed frequent graph patterns. In *ACM SIGKDD, washington, dc, usa, august 24 - 27, 2003*, 2003, pages 286–295.

[RGN08] Luc De Raedt, Tias Guns, and Siegfried Nijssen. Constraint programming for itemset mining. In *ACM SIGKDD*, 2008, pages 204–212.

[NDGN13] Benjamin Négrevergne, Anton Dries, Tias Guns, and Siegfried Nijssen. Dominance programming for itemset mining. In *IEEE ICDM*, 2013, pages 557–566.

[AGS16] John O. R. Aoga, Tias Guns, and Pierre Schaus. An efficient algorithm for mining frequent sequence with constraint programming. In *ECML PKDD*, 2016, pages 315–330.

[GGQRS] Martin Gebser, Thomas Guyet, René Quiniou, Javier Romero, and Torsten Schaub. Knowledge-based sequence mining with ASP. In *IJCAI, ny, usa, 2016*, pages 1497–1504.

[BHPW10] Mario Boley, Tamás Horváth, Axel Poigné, and Stefan Wrobel. Listing closed sets of strongly accessible set systems with applications to data mining. *Theor. comput. sci.*, 411(3):691–700, 2010.

[AU] Hiroki Arimura and Takeaki Uno. *Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems*. In. *Proceedings of the 2009 siam international conference on data mining*, pages 1088–1099.

[BL06] Francesco Bonchi and Claudio Lucchese. On condensed representations of constrained frequent patterns. *Knowledge and information systems*, 9(2):180–201, 2006.

[IM96] Tomasz Imielinski and Heikki Mannila. A database perspective on knowledge discovery. *Commun. acm*, 39(11):58–64, November 1996. ISSN: 0001-0782.

[IV99] Tomasz Imieliński and Aashu Virmani. Msql: a query language for database mining. *Data mining and knowledge discovery*, 3(4):373–408, 1999.

[BCFGPR] Hendrik Blockeel, Toon Calders, Élisa Fromont, Bart Goethals, Adriana Prado, and Céline Robardet. An inductive database system based on virtual mining views. In *Dmkd 2012*.

[RJLM02] Luc De Raedt, Manfred Jaeger, Sau Dan Lee, and Heikki Mannila. A theory of inductive query answering. In, *Icdm*, 2002.

[MT97] Heikki Mannila and Hannu Toivonen. Levelwise search and borders of theories in knowledge discovery. *Dmkd*, 1(3):241–258, 1997.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Acm stoc*. New York, NY, USA, 1971, pages 151–158.

[YH02] Xifeng Yan and Jiawei Han. Gspan: graph-based substructure pattern mining. In *IEEE ICDM, 9-12 december 2002, maebashi city, japan*, 2002, pages 721–724.

[PLDR15] Sergey Paramonov, Matthijs van Leeuwen, Marc Denecker, and Luc De Raedt. An exercise in declarative modeling for relational query mining. In *ILP*, 2015, pages 166–182.

[NG] Benjamin Négrevergne and Tias Guns. Constraint-based sequence mining using constraint programming. In *CPAIOR 2015, barcelona, spain, 2015*, pages 288–305.

[GDTNR13] Tias Guns, Anton Dries, Guido Tack, Siegfried Nijssen, and Luc De Raedt. Miningzinc: A modeling language for constraint-based mining. In *IJCAI*, 2013, pages 1365–1372.

[GPN16] Tias Guns, Sergey Paramonov, and Benjamin Négrevergne. On declarative modeling of structured pattern mining. In *Declarative learning based programming, AAAI*, 2016.

[CHSZ08] Vineet Chaoji, Mohammad Al Hasan, Saeed Salem, and Mohammed J. Zaki. An integrated, generic approach to pattern mining: data mining template library. *Data mining and knowledge discovery*, 17(3):457–495, 2008.

[ZPOL97] Mohammed J Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. Technical report. Rochester, NY, USA, 1997.

[GGM04] Floris Geerts, Bart Goethals, and Taneli Mielikäinen. Tiling databases. In *Ds*, 2004.

[VLS11] Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Dmkd*, 23(1):169–214, 2011.

[BKS10] Tijl De Bie, Kleanthis-Nikolaos Kontonasios, and Eirini Spyropoulou. A framework for mining interesting pattern sets. *SIGKDD explorations*, 12(2):92–100, 2010.