Closed Frequent Itemset Mining with Arbitrary Side Constraints

Gökberk Koçak*, Özgür Akgün*, Ian Miguel*, Peter Nightingale†
*School of Computer Science, University of St Andrews, St Andrews, Fife KY16 9SX, UK
{gk34, ozgur.akgun, ijm}@st-andrews.ac.uk
†Department of Computer Science, University of York, Deramore Lane, Heslington, York YO10 5GH, UK
peter.nightingale@york.ac.uk

***Abstract*—Frequent itemset mining (FIM) is a method for finding regularities in transaction databases. It has several application areas, such as market basket analysis, genome analysis, and drug design. Finding frequent itemsets allows further analysis to focus on a small subset of the data. For large datasets the number of frequent itemsets can also be very large, defeating their purpose. Therefore, several extensions to FIM have been studied, such as adding high-utility (or low-cost) constraints and only finding closed (or maximal) frequent itemsets. This paper presents a constraint programming based approach that combines arbitrary side constraints with closed frequent itemset mining. Our approach allows arbitrary side constraints to be expressed in a high level and declarative language which is then translated automatically for efficient solution by a SAT solver. We compare our approach with state-of-the-art algorithms via the MiningZinc system (where possible) and show significant contributions in terms of performance and applicability.**

***Index Terms*—Data mining, Pattern mining, Frequent itemset mining, Closed frequent itemset mining, Constraint Modelling**

I. Introduction

Frequent itemset mining (FIM) finds regularities in transaction databases. It has numerous applications, such as to market basket analysis, genome analysis, and drug design [1], [2]. Finding frequent itemsets allows further analysis and human inspection to focus on a small subset of the data [3].

FIM is performed on transaction databases, where each transaction is a set of items. For a subset of items S, we define $support(S)$ to represent the number of transactions that have S as a subset. A frequent itemset is any S with $support(S) \geq t$, where t is the threshold of frequency. This threshold is often given as a percentage of the total number of transactions in the dataset.

The number of transactions and the number of items in a single transaction may vary greatly among application domains. For example, among the sixteen datasets listed on the CP4IM website[1] and hosted on the UCI Machine Learning Repository [4], the number of transactions range between 101 and 8124, and the number of items ranges between 27 and 287.

Lymphography [5] is a dataset with just 148 transactions and 68 items. Despite this relatively small size, there are nearly ten million frequent itemsets in this dataset (with $t = 10\%$). Extensions to FIM were proposed to reduce the number of frequent itemsets and to produce more focused results. There are broadly two categories of these extensions in the literature:

1) application-specific side constraints on the frequent itemsets;
2) constraints between solutions, such as maximality or closedness.

The combination of these two kinds of constraints was explored in [6], where the employed algorithm needs to be carefully configured depending on the properties of the side constraints. In this paper, we present a constraint programming based declarative approach that works with arbitrary side constraints completely automatically. We focus on closed (instead of maximal) frequent itemset mining since it is a significantly more difficult problem to solve. Our approach can be applied to maximal frequent itemset mining with a minor change to a specification. No programming whatsoever is required.

Figure 1 presents an Essence [7]–[9] specification for closed and frequent itemset mining, which we will use

```
language Essence 1.3
letting ITEM be domain int(...)
letting SUPPORT be domain int(...)
given db : mset of set of ITEM
given min_support : int
given current_size : int
given solutions_so_far :
  set of (set of ITEM, SUPPORT)
find fis :
  (set (size current_size) of ITEM, SUPPORT)
such that
  fis[2] = (sum entry in db .
              toInt(fis[1] subsetEq entry)),
  fis[2] >= min_support,
  C(fis),
  forAll (sol, sup) in solutions_so_far .
    (fis[1] subset sol) -> (fis[2] > sup)
```

Fig. 1. Essence specification for Closed and Constrained Frequent Itemset Mining, slightly abbreviated.

This work was supported via EPSRC EP/P015638/1.
[1]https://dtai.cs.kuleuven.be/CP4IM/datasets/

throughout this paper. ESSENCE is a constraint specification language whose key feature is support for abstract decision variables, such as multisets, sets, functions, and relations, as well as nested types, such as multisets of relations, or the set of tuples present in the figure. The abstraction of ESSENCE enables novel solving strategies [10], [11]. ESSENCE specifications may be solved via a toolchain comprising the CONJURE [12]–[14] and SAVILE ROW [15]–[18] automated constraint modelling tools. SAVILE ROW has multiple backends to accommodate solution via constraint or SAT solvers. Herein we employ the SAT solver `nbc_minisat_all` [19].

In the specification, lines 4–5 contain the declarations for the initial database represented as a multi-set of sets, and the `min_support` value that is the frequency threshold. Key to our approach is the iterative solution of the model refined from this specification, starting with the largest frequent itemset cardinality and decrementing the cardinality by one at each iteration. The cardinality of the frequent itemset at each iteration is governed by the `current_size` parameter at Line 6. Lines 7–8 define an additional parameter needed for iteratively solving this model, as detailed in Section IV. Lines 9–10 contain the decision variable `fis`, whose domain is a tuple with two components: the set of items, and their support. Lines 12–14 post the frequency constraint, and Line 15 represents a placeholder for the side constraint. Our method supports any constraint at this position, and in this paper we focus on two side constraints from the literature: high utility [20] and low cost [6]. Lines 16–17 represent the closedness constraint. This constraint can be straightforwardly modified to find maximal frequent itemsets instead.

A. Contributions

We give a high-level declarative problem specification in ESSENCE for the closed frequent itemset mining problem and a method that supports arbitrary side constraints. Our method is completely automated: it does not require the user to make any decisions about which mining algorithm to employ. We present an exhaustive empirical study where we compare our method to several execution plans offered by MiningZinc [20], [21]. We also construct non-trivial instances for the high-utility and low-cost closed itemset mining problems at varying levels of frequency thresholds, which we hope will become valuable to facilitate further research on these problems.

II. BACKGROUND: CONSTRAINT PROGRAMMING AND PROPOSITIONAL SATISFIABILITY

In this paper we use existing tools for solving discrete decision-making and optimisation problems. It is natural to characterise such problems as a set of decision variables, each representing a choice that must be made in order to solve the problem at hand (e.g. which staff member is on duty for the Friday night shift), and a set of constraints describing allowed combinations of variable assignments (e.g. a staff member cannot be assigned to a day shift immediately following a night shift). A solution is an assignment of a value to each variable satisfying all constraints. Many decision-making and optimisation formalisms take this general form, including: constraint programming (CP) [22], propositional satisfiability (SAT) and its extensions [23], and operations research approaches, particularly Mixed Integer Programming (MIP) [24]. These approaches have much in common, but differ in the types of decision variables and constraints they support, and the inference mechanisms used to find solutions.

The Propositional Satisfiability Problem (SAT) is to find an assignment to a set of Boolean variables so as to satisfy a given Boolean formula, typically expressed in conjunctive normal form [23]. SAT has many important applications, such as hardware design and verification, planning, and combinatorial design [25]. Powerful, robust solvers have been developed for SAT employing techniques such as conflict-driven learning, watched literals, restarts and dynamic heuristics for backtracking solvers [26], and sophisticated incomplete techniques such as stochastic local search [27]. We employ an AllSAT solver [19] designed to find all solutions to a given Boolean formula. The AllSAT solver we use (`nbc_minisat_all`) is a backtracking solver benefiting from conflict-driven learning and the other techniques mentioned above.

CP provides a richer language of discrete variables with domains either given in extension or expressed in terms of upper and lower bounds, arithmetic and logical operators over these variables, and a library of 'global' constraints that capture common reasoning patterns. We use the automated modelling assistant SAVILE ROW [15]–[17] that takes a solver-independent CP model in the language ESSENCE PRIME and translates it into a form suitable for a specific solver, while also improving the formulation using a variety of techniques. SAVILE ROW supports integer and Boolean decision variables, and matrices of these two types. It uses the MINION [28] CP solver in a preprocessing mode to filter the variable domains. In this paper we use SAVILE ROW to encode the problem instances into SAT before solving with the AllSAT solver.

CONJURE [12]–[14] is an automated constraint modelling tool, supporting higher-level variable types such as partitions, sets, multisets, functions and sequences (nested arbitrarily) expressed in the ESSENCE language [7]–[9]. CONJURE translates ESSENCE specifications into ESSENCE PRIME suitable for input into SAVILE ROW. Our use of CONJURE enables natural, declarative and very concise specifications of frequent itemset mining and its variations.

III. EXTENSIONS TO FREQUENT ITEMSET MINING

Plain frequent itemset mining is concerned only with finding subsets of items that occur together in a transaction database. There are typically a large number of frequent itemsets and algorithms like Apriori, LCM and Eclat provide very efficient ways of enumerating them. For example, for the dataset in Figure 2 with four items (in $\mathbb{I}$) and three transactions (in $\mathbb{T}$), there are ten frequent itemsets if the minimum support is 2.

$$\mathbb{I} = \{1, 2, 3, 4\}$$

$$\mathbb{T} = \{\{1, 2, 4\}, \{1, 2, 3, 4\}, \{3, 4\}\}$$

$$FIS = \{\{\},\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \\ \{1, 4\}, \{2, 4\}, \{3, 4\}, \{1, 2, 4\}\}$$

Fig. 2. A small database of transactions

A. Side constraints

Application-specific side constraints are often added to an itemset mining problem to provide focus for the results. Some of these constraints are simple: excluding a certain itemset altogether, insisting that a certain itemset has to be in every result, putting lower/upper bounds on the cardinality of itemsets, etc. Constraints like these have been incorporated into existing frequent itemset solvers in the past. For example, LCMv2 [29] does not support itemset cardinality constraints, however a later version (LCMv5) added support for them. Adding support for new side constraints inside a dedicated itemset solver requires modification to the solver. Moreover, every new constraint requires non-trivial reasoning in terms of its integration to the existing algorithm.

High-utility frequent itemset mining is a more complex extension. Recent work [20] presents a constraint model for this problem, which is a straightforward encoding of a utility value per item and an arithmetic constraint requiring the sum of utilities in an itemset to be greater than a minimum utility value. In our running example in Figure 2 if we use $1, 2, 2, 1$ as the utility values for items $1, 2, 3, 4$ respectively, and with a minimum utility threshold of 3, the remaining frequent itemsets are $\{\{1, 2\}, \{2, 4\}, \{3, 4\}, \{1, 2, 4\}\}$.

Low-cost frequent itemset mining is another similar extension that is used as a motivating example in [6]. In low-cost itemset mining we post an arithmetic constraint that requires the sum of all costs in an itemset to be lower than a cost threshold, as opposed to greater than a utility threshold in high-utility mining. In our running example, using $1, 2, 2, 1$ as cost values for items $1, 2, 3, 4$ respectively, and with a maximum cost threshold of 3, the remaining frequent itemsets are $\{\{\}, \{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 4\}, \{2, 4\}, \{3, 4\}\}$.

B. Constraints among solutions

Another class of extensions to frequent itemset mining involves constraints among solutions, which are written in terms of the full set of solutions (all frequent itemsets). Maximality is a well-known constraint in this class [30]. A frequent itemset is maximal if and only if none of its supersets are frequent. This condition intuitively follows from the observation that if an itemset is frequent all of its subsets are also frequent. Including them in the results does not add value.

Closedness is a similar constraint [31]. A frequent itemset is closed if and only if its support is greater than that of all of its supersets. Hence, in contrast to maximal itemset mining, the result set of closed itemset mining may include subsets of other closed itemsets if the support of the smaller set is greater than the support of the larger set. Intuitively, the closed itemsets are a succinct representation of the full set of frequent itemsets, whereas the maximal itemsets are not.

In our running example with `min_support` 2, the maximal itemsets are $\{\{3, 4\}, \{1, 2, 4\}\}$ and the closed itemsets are $\{\{4\}, \{3, 4\}, \{1, 2, 4\}\}$. This is because $\{4\}$ has support of 3, which is greater than that of all of its frequent supersets.

Herein we focus on closed frequent itemset mining since it is a significantly more difficult problem to solve. Our approach can be applied to maximal frequent itemset mining with a small change to the ESSENCE specification and no programming whatsoever.

C. Closed and constrained itemset mining

Combining problem specific side constraints with constraints between solutions is appealing since this combination would provide the benefits of both classes of extensions. There is some ambiguity about what this combination might mean and this was one of the motivations of our method.

There are two possible definitions for the combined problem: (1) all closed frequent itemsets that also satisfy the side constraint (2) all frequent itemsets that satisfy the side constraint and are closed within this solution set.

The former is strictly less useful since when we remove a closed frequent itemset from the solution set due to the side constraint, we might also remove several of its subsets. We demonstrate the difference between the two definitions in Table I using our running example with a minimum support value of 2, and a minimum cost threshold of 3. Starting from the same database, the two methods reach different sets of solutions. In the first table, the problem arises from removing the set $\{1, 2, 4\}$ and consequently losing all of its subsets from the solution set. This produces an incomplete set of solutions.

This problem only occurs when the side constraint is not monotone [6]. A side constraint C is monotone when for any two frequent itemsets a and b with $a \subset b$, $C(a) \implies C(b)$. The two side constraints that we consider in this paper are high-utility (monotone) and low-cost (not monotone).

The combination of these two kinds of constraints was explored in [6], where the employed algorithm needs to be carefully configured depending on properties of the side constraints. In this paper, we present a constraint programming based declarative approach that works with arbitrary side constraints completely automatically.

IV. CLOSED FIM WITH ARBITRARY SIDE CONSTRAINTS

Our approach to closed itemset mining (with or without side constraints) is iterative. We first find frequent itemsets of the largest cardinality possible, that of the largest transaction. For the largest cardinality, all frequent itemsets are guaranteed to be also closed frequent itemsets, since they have no frequent supersets that can rule them outside of the closed set. We then iteratively decrement the cardinality by one and solve for all frequent itemsets. At each iteration we also produce an *exclusion constraint* (see Lines 13–14 of Figure 1), which

TABLE I
THE DIFFERENCE BETWEEN THE ORDER OF APPLICATION OF SIDE CONSTRAINTS AND THE CLOSEDNESS CONSTRAINT.

(Step 1) Database	(Step 2) Closed Itemsets	(Step 3) Closed and Low-Cost
$\{\{1,2,4\}, \{1,2,3,4\}, \{3,4\}\}$	$\{\{4\}, \{3,4\}, \{1,2,4\}\}$	$\{\{4\}, \{3,4\}\}$

(Step 1) Database	(Step 2) Frequent Itemsets	(Step 3) Low-Cost	(Step 4) Low-Cost and Closed
$\{\{1,2,4\}, \{1,2,3,4\}, \{3,4\}\}$	$\{\{\}, \{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,4\}, \{2,4\}, \{3,4\}, \{1,2,4\}\}$	$\{\{\}, \{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}, \{1,4\}, \{2,4\}, \{3,4\}\}$	$\{\{4\}, \{1,2\}, \{1,4\}, \{2,4\}, \{3,4\}\}$

ensures that all solutions found at a certain iteration are closed with respect to solutions found at previous iterations. This approach is sound, i.e. we do not produce any frequent itemsets that are not closed, since the closedness property of an itemset only depends on larger frequent itemsets. It is also complete since we enumerate all solutions for every itemset cardinality. This is related to general constraint dominance programming approaches [32], [33]. Here, we present a specialised method that exploits the dominance relationship among the solutions and the order of search to achieve high-performance.

```
1:  procedure ITERATIVEMINER(D)
2:      $ub \leftarrow \text{SMARTSTART}(D)$
3:      $solutions \leftarrow \{\}$
4:      $size \leftarrow ub$
5:      while $size \geq 0$ do
6:          $s \leftarrow \text{SOLVE}(D, solutions, size)$
7:          $solutions \leftarrow solutions \cup s$
8:          $size \leftarrow size - 1$
9:      end while
10:     return $solutions$
11: end procedure
```

Fig. 3. Iterative Closed and Constrained Itemset Mining with CP

For a given problem class CONJURE is called once to refine the ESSENCE specification in Figure 1 into a constraint model in ESSENCE PRIME. CONJURE produces an Occurrence model [34] for the set variable, which uses an array of Boolean decision variables for each item. SAVILE ROW employs the MINION constraint solver as a preprocessing step that achieves singleton arc consistency on the lower and upper bounds of decision variables. This step has the potential to reduce the number of iterations considerably by reducing the range of itemset cardinalities that need to be considered. Thereafter, each iteration is performed as described in Figure 3.

The SMARTSTART procedure that is called on Line 2 of Figure 3 invokes Eclat [35] to get a good upper bound. Eclat is a frequent itemset solver that does not support any of the side constraints that we are interested in. For some datasets Eclat is able to count the number of maximal frequent itemsets at each cardinality level very quickly. For others it takes a very long time and/or a lot of memory. We use a simple method of benefiting from Eclat at this step: we run it with a fixed time limit of three minutes and a memory limit of 15GB. If it finishes within these limits, we use the cardinality of the largest maximal frequent itemset as our starting point. Otherwise, we start from the cardinality of the widest transaction in the database. In our experiments this SMARTSTART phase often helps us reduce the number of iterations.

The main loop (lines 5–9) is implemented within a lightly modified version of SAVILE ROW. The problem instance is translated into SAT using the standard encoding provided by SAVILE ROW. We selected a SAT solver due to the large number of Boolean variables in the model and the relatively simple constraints. We use `nbc_minisat_all` [19], which is an AllSAT solver based on MiniSat [36]. AllSAT solvers are designed to enumerate all solutions of an instance of SAT. The SAT encoding is generated once then incrementally modified (by adding new exclusion constraints and changing the value of the *size* variable) for each iteration of the main loop.

V. EMPIRICAL EVALUATION

We test our method on all of the sixteen datasets listed on the CP4IM website. We design two closed itemset mining experiments with side constraints and compare the performance of our method against MiningZinc. MiningZinc works with a declarative model, and calculates a number of execution plans after inspecting a given model. It is hard to estimate the relative performance of these execution plans, so in the interest of fairness we ran every execution plan offered by MiningZinc.

The first experiment uses a high-utility constraint, and we compare our computational results against the MiningZinc model given in Figure 8 of [20]. The second experiment uses a low-cost constraint in addition to the high-utility constraint. The low-cost constraint can be written in the MiningZinc language in a similar way to the high-utility constraint in the model of Figure 8 of [20]. MiningZinc produces a number of execution plans when provided with this model. However, all of these execution plans produce faulty answers. The low-cost constraint is not monotonic, so the MiningZinc execution plans suffer from the problem we describe in Section III-C.

Experiments were performed with 16 processes in parallel on a 32-core AMD Opteron 6272 at 2.1 GHz with 256 GB RAM. We modified the MiningZinc source code to use a different temporary directory for each of its invocation. By default MiningZinc uses a fixed directory for its temporary files, which precludes us from running multiple MiningZinc processes at the same time.[2]

A. *Closed High-Utility Itemset Mining*

In order to experiment with high-utility itemset mining, we need to assign utility values for each item and decide on a threshold value for total utility. We do this by uniformly randomly assigning a value between 0 and 5 to each item. We then choose a utility threshold value that results in finding at most tens of thousands of closed frequent itemsets. The changes to the ESSENCE specification given in Figure 1 are minimal. We add two parameters and a single arithmetic constraint as seen below. We experiment with the same frequency thresholds (10%, 20%, 30%, 40% and 50%) as [20].

```
given utilities : matrix indexed by
 [ITEM] of int(0..5)
given min_util : int
such that
 (sum item in fis[1] . utilities[item])
  >= min_util
```

MiningZinc calculates eighteen execution plans for closed high-utility itemset mining. Plans 1–2 run Eclat to find closed itemsets, followed by different ways of filtering the closed itemsets using a constraint programming based approach. Plans 3–6 run a preprocessing step followed by different configurations of Gecode [37]. Plans 7–14 run Gecode without a preprocessing step. Plans 15–18 run LCM (v2 and v5) to find closed itemsets, followed by a constraint programming based approach to filter the side constraints.

All of these execution plans either calculate closed itemsets first and then apply side constraints, or contain the closedness constraint inside a constraint model and apply it simultaneously with the side constraints.

For sixteen datasets and five different frequency levels we construct 80 instances. With a 3-hour time limit, we are able to solve all but 6 of these instances. We also run all execution plans produced by MiningZinc, and 12 plans out of 18 produce an incorrect number of solutions for at least one instance. To be clear, this is not for reasons of monotonicity, since the high-utility constraint is monotonic; the reasons are unknown to us. We exclude these execution plans from our comparison.

Table II gives the runtimes of several MiningZinc execution plans and our method (ESSENCE Mining). The last line of each heading indicates the execution plan we use. For example `MZ-11` means that this was the eleventh execution plan. In square brackets we indicate the Gecode option used, where F represents a model rewriting called `freq_items`, and R represents another called `reify`. FI indicates a preprocessing step, `FreqItems`, that runs before Gecode.

The ESSENCE Mining column contains time taken by our method, including modelling overhead and the call to Eclat (inside the SMARTSTART procedure). We compare this runtime against the runtimes of the MiningZinc execution plans and indicate the winner using a bold font. In addition, we provide the time taken by only the AllSAT solver for our method in the ESSENCE Mining (Solver Time) column. In general, the modelling overhead is small. However in some cases (for example in the audiology dataset) there is a significant difference between the total time and the solver time. This is a unique case in our experiments: on the audiology dataset the SMARTSTART procedure always times out (we use a fixed time limit of 3 minutes) without making any progress. On the harder instances of the `hypothyroid` dataset our method is the only one that finishes before the time limit.

Overall, on this experiment our method, which does not require the user to select among a large set of execution plans, reliably finds correct results and is competitive with the six of the eighteen execution plans produced by MiningZinc that produce correct solutions. This is despite not exploiting the monotonicity of the high-utility constraint. However, the greatest benefit of our approach is its generality. In the next section we analyse our performance in the presence of a constraint that is not monotone.

B. *Closed High-Utility and Low-Cost Itemset Mining*

We experiment with a combined high-utility and low-cost itemset mining problem. To generate costs per item and a cost threshold, we follow a similar procedure to that of generating utility values and the utility threshold. The cost values are uniformly randomly chosen to a value between 0 and 5. We also maintain the utility values that were previously generated. A cost threshold and a utility threshold is chosen to limit the number of closed frequent itemsets to at most tens of thousands of closed frequent itemsets. We add two more parameters and another arithmetic constraint.

```
given cost_values : matrix indexed by
 [ITEM] of int(0..5)
given max_cost : int
such that
 (sum item in fis[1] . cost_values[item])
  <= max_cost
```

Extending our ESSENCE problem specification to work with the low-cost constraint is trivial. We add the necessary statements for the parameter values and post an arithmetic constraint that requires the total cost of an itemset to be less than the cost threshold.

We also tried extending the MiningZinc model (from Figure 8 of [20]) similarly. Due to the issue that was explained in Section III-C the extended model gives incorrect results: it misses a large number of solutions. Hence, comparing our performance to this model is not sensible.

[2]We make all of our data files and software available at http://github.com/stacs-cp/OEDM18-mining

TABLE II
CLOSED HIGH-UTILITY ITEMSET MINING RESULTS ON 9 DATASETS. TIMES ARE IN SECONDS (* INDICATES A 3-HOUR TIMEOUT).

Instance	Gecode (MZ-11)	Gecode [F] (MZ-13)	Gecode [FR] (MZ-12)	Gecode [R] (MZ-14)	FI + Gecode [F] (MZ-6)	FI + Gecode [FR] (MZ-5)	Essence Mining	Essence Mining (Solver Time)	Number of Solutions
lymph-50	**2**	**2**	3	3	3	3	15	0	684
lymph-40	**3**	**3**	4	4	**3**	4	17	1	981
lymph-30	**4**	**4**	6	5	**4**	6	16	2	1,042
lymph-20	**6**	**6**	8	8	**6**	8	22	7	1,844
lymph-10	**13**	**13**	19	19	**13**	19	48	29	4,443
krvskp-50	704	702	149	150	710	**145**	964	842	1
krvskp-40	1,008	1,002	253	**249**	1,006	**249**	1,621	1,487	5
krvskp-30	4,674	4,692	**626**	629	4,678	629	3,883	3,732	18
krvskp-20	*	*	1,651	**1,636**	*	1,651	10,171	9,985	113
krvskp-10	*	*	3,708	**3,657**	*	3,697	10,746	10,509	189
hypo-50	*	*	643	**624**	*	632	3,226	3,076	302
hypo-40	*	*	2,027	**2,019**	*	2,020	8,045	7,872	691
hypo-30	*	*	6,933	**6,842**	*	6,923	10,111	9,911	163
hypo-20	*	*	*	*	*	*	**6,936**	6,711	16
hypo-10	*	*	*	*	*	*	**10,646**	10,422	32
hepatitis-50	5	5	**4**	**4**	5	5	15	2	444
hepatitis-40	13	13	**7**	**7**	13	**7**	17	5	251
hepatitis-30	42	41	**19**	**19**	42	**19**	33	19	659
hepatitis-20	51	52	**29**	**29**	52	**29**	58	32	555
hepatitis-10	99	106	98	**94**	108	99	247	204	11,493
heart-50	13	13	**12**	**12**	13	**12**	25	8	29
heart-40	**36**	37	**36**	**36**	37	**36**	51	33	27
heart-30	197	201	**116**	117	198	117	254	218	47
heart-20	901	904	496	**491**	906	497	1,039	997	635
heart-10	3,971	3,991	3,387	3,383	3,971	**3,374**	*	*	38,774
german-50	27	28	**13**	**13**	27	**13**	59	19	8
german-40	49	48	26	**25**	48	**25**	125	85	5
german-30	141	148	52	**51**	147	52	524	469	2
german-20	732	714	214	216	716	**212**	1,295	1,234	53
german-10	6,063	6,151	2,011	2,021	6,071	**2,008**	6,701	6,588	5,911
australian-50	58	59	31	32	58	**30**	87	56	2
australian-40	245	242	132	130	240	**129**	636	582	104
australian-30	892	878	380	**376**	870	381	1,791	1,732	7
australian-20	3,848	3,812	1,531	**1,510**	3,825	1,514	6,409	6,342	146
australian-10	*	*	**6,901**	6,959	*	6,962	*	*	717
audiology-50	2,098	2,111	241	**239**	2,146	242	286	48	166
audiology-40	1,362	1,360	226	**225**	1,370	**225**	364	48	3,463
audiology-30	649	648	**196**	197	648	198	260	43	2,278
audiology-20	307	312	**209**	211	311	211	235	51	738
audiology-10	215	**214**	282	277	215	280	329	122	7,643
anneal-50	75	75	17	**16**	75	**16**	63	28	39
anneal-40	124	124	43	43	123	**42**	129	84	63
anneal-30	266	270	99	**98**	274	**98**	293	244	163
anneal-20	514	514	**277**	283	512	**277**	654	609	893
anneal-10	774	**771**	1,102	1,105	786	1,094	2,688	2,601	18,335

Following the analysis of [6] we decided to relax the closedness condition for MiningZinc and find all frequent itemsets that satisfy the side constraints. This is the only sensible comparison since the full set of closed frequent itemsets is a lossless compression of the full set of frequent itemsets, and thus the two contain identical amounts of information. In other words, we use MiningZinc to perform the first three steps in the second table of Table I. To achieve the same results as our method another procedure would be needed in MiningZinc.

We verified the correctness of our results compared with those obtained by MiningZinc by implementing an optional post-processing step that expands the set of closed frequent itemsets into the set of all frequent itemsets satisfying the side constraints. We found the same number of frequent itemsets as MiningZinc whenever ESSENCE Mining did not time out.

Table III contains the results of this experiment. We limited this experiment to the six execution plans that gave correct results in Section V-A. Often the number of closed frequent itemsets are much smaller than the number of all frequent itemsets, therefore directly calculating the set of closed itemsets may have less overhead. However it is important to note that from the full set of closed itemsets (our results), it is

TABLE III
Closed High-Utility and Low-Cost Itemset Mining on 9 datasets. Times are in seconds (* indicates a 3-hour timeout).

Instance	Gecode (MZ-11)	Gecode [F] (MZ-13)	Gecode [FR] (MZ-12)	Gecode [R] (MZ-14)	FI + Gecode [F] (MZ-6)	FI + Gecode [FR] (MZ-5)	Essence Mining	Essence Mining (Solver Time)	Number of Solutions
lymph-50	15	**5**	10	10	**5**	9	16	1	1,492
lymph-40	33	**8**	23	23	9	20	17	2	1,711
lymph-30	103	29	93	91	32	80	**24**	6	3,902
lymph-20	194	60	202	203	64	188	**38**	17	6,322
lymph-10	920	526	987	1,263	447	1,002	**165**	126	22,606
krvskp-50	*	3,896	282	281	3,911	**274**	1,001	872	381
krvskp-40	*	6,774	794	792	6,901	**761**	2,221	2,079	1,035
krvskp-30	*	*	3,057	3,057	*	**2,971**	4,735	4,531	4,371
krvskp-20	*	*	*	*	*	*	*	*	*
krvskp-10	*	*	*	*	*	*	*	*	*
hypo-50	*	*	*	*	*	*	**4,984**	4,498	30,950
hypo-40	*	*	*	*	*	*	**2,455**	2,281	1,629
hypo-30	*	*	*	*	*	*	**7,800**	7,574	2,748
hypo-20	*	*	*	*	*	*	**5,585**	5,358	39
hypo-10	*	*	*	*	*	*	**6,036**	5,727	481
hepatitis-50	15	**9**	15	15	**9**	15	21	6	3,590
hepatitis-40	67	**37**	72	73	39	68	54	31	12,587
hepatitis-30	322	195	256	254	200	237	**83**	54	18,139
hepatitis-20	1,862	1,402	1,192	1,191	1,452	1,119	**214**	175	23,379
hepatitis-10	10,368	8,614	3,283	3,356	9,457	2,962	**251**	211	17,685
heart-50	55	**16**	18	17	**16**	**16**	27	8	483
heart-40	254	107	122	120	110	113	**92**	68	3,630
heart-30	1,469	760	521	516	774	485	**298**	264	7,165
heart-20	*	8,219	7,561	7,585	8,082	6,564	**3,367**	3,224	68,040
heart-10	*	*	*	*	*	*	**4,380**	4,296	19,053
german-50	98	36	**23**	**23**	35	**23**	72	27	377
german-40	354	136	127	128	140	**120**	182	125	1,969
german-30	1,479	662	893	883	671	844	**620**	519	9,087
german-20	7,049	3,532	6,371	6,369	3,618	5,683	**1,904**	1,614	40,311
german-10	*	*	6,780	7,288	*	6,485	**4,730**	4,633	5,581
australian-50	479	200	182	177	204	177	**166**	118	2,093
australian-40	2,860	1,688	1,182	1,174	1,764	1,094	**553**	492	4,474
australian-30	*	*	10,672	10,647	*	10,272	**2,131**	2,003	16,020
australian-20	*	*	*	*	*	*	**9,044**	8,862	22,374
australian-10	*	*	*	*	*	*	*	*	*
audiology-50	*	*	1,123	1,115	*	1,025	**265**	17	6,699
audiology-40	*	*	436	433	*	370	**274**	14	8,283
audiology-30	*	4,688	423	417	4,751	349	**259**	14	7,357
audiology-20	*	*	1,940	1,950	*	1,647	**260**	28	9,667
audiology-10	*	*	5,081	5,113	*	4,591	**254**	33	6,761
anneal-50	*	*	*	*	*	*	**411**	335	17,456
anneal-40	*	*	*	*	*	*	**1,595**	1,474	36,041
anneal-30	*	*	*	*	*	*	**1,150**	1,048	25,487
anneal-20	*	*	*	*	*	*	**1,672**	1,572	24,728
anneal-10	*	*	*	*	*	*	**2,809**	2,683	32,689

trivial to generate the set of all frequent itemsets.

Lymphography (`lymph`) is a relatively small that has 10M frequent itemsets and 47K closed itemsets (at 10% frequency). The results show that there is an overhead in our system for easy instances, for `lymph-50` even though the solver spends 1 second, the total time is 16 seconds. This overhead quickly becomes negligible for harder instances, in the rest of the `lymph-X` instances the speed-up achieved by our method progressively increases as the instance becomes harder.

Kr-vs-kp (`krvskp`) is the only instance where our method is consistently slower than MiningZinc. This dataset is both very large and it also seems to have a very large ratio between the number of frequent itemsets and closed frequent itemsets. Intuitively the set of frequent itemsets of this dataset does not compress very well. Since the number of closed itemsets is very close to the number of frequent itemsets, it is not surprising that a more specialised tool that focuses on mining tasks handles this case slightly better.

Hypothyroid (`hypo`) is a medical dataset that is of a similar size to Kr-vs-kp, but seems to have a lower ratio of closed itemsets to frequent itemsets. Our approach is the only one that can complete the task in the 3-hour time limit. The performance does not seem to directly depend on the number

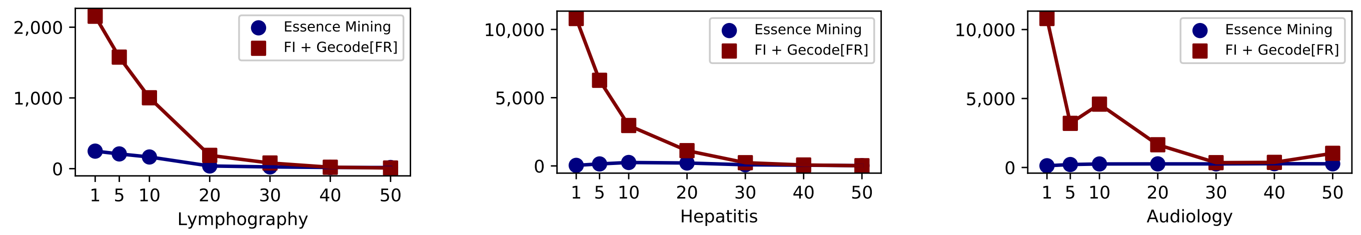

Fig. 4. Closed High-Utility and Low-Cost itemset mining for Lymphography, Hepatitis, and Audiology datasets at lower levels of frequency thresholds. The horizontal axes contain the frequency level that we use, and the vertical axis is time in seconds.

of solutions or the frequency threshold.

Hepatitis is another medical dataset that has a low ratio of frequent itemsets to close itemsets (less than 0.01%). We observe a similar phenomenon here to Lymphography. As the instances get harder (and the frequency threshold gets lower) our approach becomes relatively much faster. We show that this behaviour does not stop at the frequency level 10% and give results for lower frequency levels in section Section V-C.

Heart-cleveland (`heart`) is a third medical dataset that is very similar to the Hepatitis dataset in its behaviour. One difference between this dataset and Hepatitis is that this datasets had around an order of magnitude more closed frequent itemsets. The performance of our method progresses similarly to Hepatitis at a slightly larger scale.

The German-credit (`german`) and Australian-credit (`australian`) datasets relate to credit risks and credit card transactions. The German dataset is smaller and has a ratio of roughly 1/20 between frequent itemsets and closed frequent itemsets. For easy instances of this dataset the modelling overhead of our method causes us to be slower than MiningZinc. For harder instances and for all instances of the Australian dataset our approach is comfortably faster.

Audiology is one of the largest datasets with more than 167M closed frequent itemsets (at 10%). We were not able to calculate the ratio of the number of frequent itemsets to the number of closed itemsets since there are a very large number of both. Our method is much faster for all five instances that we tested. Noticeably, the our solver time is tiny in comparison to the total time, this is due to running Eclat until the 3-minute time limit without any gain. Improving the SMARTSTART procedure to avoid cases like this is an important future research direction.

The Anneal dataset relates to steel annealing. It has a very small ratio of frequent itemsets to closed frequent itemsets and hence is a prime opportunity for our method to shine. For all instances of this dataset, all execution plans offered by MiningZinc reach the time limit whereas our method finishes the task in less than one third of the time limit.

In all of our experiments we sometimes find a trade-off between different execution plans offered by MiningZinc. No single execution plan seems to give the best results for all instances. For example, even though `FI + Gecode[FR]` is faster in a large number of cases, `FI + Gecode[F]` is faster for all instances of the Lymphography dataset (see Table III).

C. *Lower frequency thresholds for a selection of datasets*

In the Lymphography, Hepatitis and Audiology datasets, our method provides the best performance for all instances by roughly one order of magnitude for the five frequency levels. In order to demonstrate the performance of our method at lower frequency levels, for these three datasets we ran additional experiments to compare our method with the `FI + Gecode[FR]` execution plan of MiningZinc (since this is the best option overall in our earlier experiments). Figure 4 contains the runtimes of the two solvers on these instances. We see that at lower frequency levels, our method presents an even bigger advantage. This is due to the decreased ratio of closed itemsets to frequent itemsets at lower frequency levels.

VI. CONCLUSION

In this paper we have presented a high-level declarative problem specification in ESSENCE for the closed frequent itemset mining problem and a method that supports arbitrary side constraints. Our method is completely declarative and automated. It does not require the user to make any decisions about which mining algorithm to employ. To the best of our knowledge this is the first declarative method of performing the closed frequent itemset mining task with arbitrary side constraints (whether they are monotone or not).

We tested our method against all execution plans offered by MiningZinc for two different kinds of side constraints: high utility and low cost. The former is monotonic and hence MiningZinc offers very efficient ways handling it. We found that even though our approach does not exploit the monotonicity of the constraint if offers competitive results. The latter constraint is not monotonic and our approach proves to be very effective in handling it.

We used all datasets published on the CP4IM website in our experiments, however these datasets do not contain utility (or cost) values and thresholds. This was a challenge: we had to create non-trivial instances that can be solved within our time limit of 3 hours. Most randomly generated threshold values (for both cost and utility) produce either trivially unsatisfiable instances or instances that are too hard. In order to aid future research in this field, we make our datasets available. We also plan to investigate a more systematic way of generating challenging instances for these problems in the near future.

Future Work

In Section V-B, we are unable to compare our results with those of MiningZinc. This is due to the missing step (Step 4 of Table I) in the MiningZinc part of our experiments. We plan to implement a compressor that takes the full set of frequent itemsets and produces the corresponding closed itemsets. Including the time required for the compression on top of the time spent by MiningZinc execution plans would also make the comparisons more fair.

In our SMARTSTART procedure, we use Eclat in its maximal frequent itemset mining mode to find the cardinality of the largest frequent itemset (without side constraints). This often helps us by reducing the number of iterations required; however, it sometimes fails to make any useful progress within the time allowed. An example is the Audiology dataset which has a very large number of maximal frequent itemsets. We plan to improve our SMARTSTART procedure to become smarter and avoid running Eclat for datasets like Audiology.

REFERENCES

[1] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD Record*, vol. 22. ACM, 1993, pp. 207–216.

[2] S. Naulaerts, P. Meysman, W. Bittremieux, T. N. Vu, W. Vanden Berghe, B. Goethals, and K. Laukens, "A primer to frequent itemset mining for bioinformatics," *Briefings in bioinformatics*, vol. 16, no. 2, pp. 216–231, 2013.

[3] P. Fournier-Viger, J. C.-W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le, "A survey of itemset mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 7, no. 4, 2017.

[4] D. Dheeru and E. Karra Taniskidou, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[5] M. Zwitter and M. Soklic, "Lymphography domain," *University Medical Center, Institute of Oncology, Ljubljana, Yugoslavia*, 1988.

[6] F. Bonchi and C. Lucchese, "On closed constrained frequent pattern mining," in *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*. IEEE, 2004, pp. 35–42.

[7] A. M. Frisch, M. Grum, C. Jefferson, B. M. Hernández, and I. Miguel, "The essence of essence," *Modelling and Reformulating Constraint Satisfaction Problems*, p. 73, 2005.

[8] A. M. Frisch, W. Harvey, C. Jefferson, B. Martínez-Hernández, and I. Miguel, "Essence: A constraint language for specifying combinatorial problems," *Constraints*, vol. 13, no. 3, pp. 268–306, 2008.

[9] A. M. Frisch, M. Grum, C. Jefferson, B. M. Hernández, and I. Miguel, "The design of essence: A constraint language for specifying combinatorial problems." in *IJCAI*, vol. 7, 2007, pp. 80–87.

[10] O. Akgün, S. Attieh, I. P. Gent, C. Jefferson, I. Miguel, P. Nightingale, A. Z. Salamon, P. Spracklen, and J. Wetter, "A framework for constraint based local search using Essence," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, 2018, pp. 1242–1248.

[11] P. Spracklen, O. Akgün, and I. Miguel, "Automatic generation and selection of streamlined constraint models via monte carlo search on a model lattice," in *Principles and Practice of Constraint Programming (CP 2018)*, 2018, pp. 362–372.

[12] Ö. Akgün, A. M. Frisch, I. P. Gent, B. S. Hussain, C. Jefferson, L. Kotthoff, I. Miguel, and P. Nightingale, "Automated symmetry breaking and model selection in conjure," in *International Conference on Principles and Practice of Constraint Programming*. Springer, 2013, pp. 107–116.

[13] Ö. Akgün, I. P. Gent, C. Jefferson, I. Miguel, and P. Nightingale, "Breaking conditional symmetry in automated constraint modelling with conjure." in *ECAI*, 2014, pp. 3–8.

[14] Ö. Akgün, I. Miguel, C. Jefferson, A. M. Frisch, and B. Hnich, "Extensible automated constraint modelling," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*. AAAI Press, 2011, pp. 4–11.

[15] P. Nightingale, Ö. Akgün, I. P. Gent, C. Jefferson, and I. Miguel, "Automatically improving constraint models in Savile Row through associative-commutative common subexpression elimination," in *20th International Conference on Principles and Practice of Constraint Programming (CP 2014)*. Springer, 2014, pp. 590–605.

[16] P. Nightingale, P. Spracklen, and I. Miguel, "Automatically improving SAT encoding of constraint problems through common subexpression elimination in Savile Row," in *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP 2015)*, 2015, pp. 330–340.

[17] P. Nightingale, Ö. Akgün, I. P. Gent, C. Jefferson, I. Miguel, and P. Spracklen, "Automatically improving constraint models in Savile Row," *Artificial Intelligence*, vol. 251, pp. 35–61, 2017.

[18] O. Akgün, I. P. Gent, C. Jefferson, I. Miguel, P. Nightingale, and A. Z. Salamon, "Automatic discovery and exploitation of promising subproblems for tabulation," in *Principles and Practice of Constraint Programming (CP 2018)*, 2018, pp. 3–12.

[19] T. Toda and T. Soh, "Implementing efficient all solutions sat solvers," *Journal of Experimental Algorithmics (JEA)*, vol. 21, pp. 1–12, 2016.

[20] T. Guns, A. Dries, S. Nijssen, G. Tack, and L. De Raedt, "Miningzinc: A declarative framework for constraint-based mining," *Artificial Intelligence*, vol. 244, pp. 6–29, 2017.

[21] L. De Raedt, T. Guns, and S. Nijssen, "Constraint programming for itemset mining," in *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2008, pp. 204–212.

[22] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*. Elsevier, 2006.

[23] A. Biere, M. Heule, and H. van Maaren, *Handbook of Satisfiability*. IOS Press, 2009, vol. 185.

[24] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, 9th ed. McGraw-Hill, 2010, international Edition.

[25] J. Marques-Silva, "Practical applications of boolean satisfiability," in *9th International Workshop on Discrete Event Systems (WODES 2008)*, 2008, pp. 74–80.

[26] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th annual Design Automation Conference*. ACM, 2001, pp. 530–535.

[27] Y. Shang and B. W. Wah, "A discrete lagrangian-based global-search method for solving satisfiability problems," *Journal of global optimization*, vol. 12, no. 1, pp. 61–99, 1998.

[28] I. P. Gent, C. Jefferson, and I. Miguel, "Minion: A fast scalable constraint solver," in *Proceedings ECAI 2006*, 2006, pp. 98–102.

[29] T. Uno, M. Kiyomi, and H. Arimura, "Lcm ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets," in *Fimi*, vol. 126, 2004.

[30] K. Gouda and M. J. Zaki, "Efficiently mining maximal frequent itemsets," in *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*. IEEE, 2001, pp. 163–170.

[31] J. Pei, J. Han, R. Mao *et al.*, "Closet: An efficient algorithm for mining frequent closed itemsets." in *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, vol. 4, 2000, pp. 21–30.

[32] B. Negrevergne, A. Dries, T. Guns, and S. Nijssen, "Dominance programming for itemset mining," in *Data Mining (ICDM), 2013 IEEE 13th International Conference on*. IEEE, 2013, pp. 557–566.

[33] T. Guns, P. J. Stuckey, and G. Tack, "Solution dominance over constraint satisfaction problems."

[34] Ö. Akgün, "Extensible automated constraint modelling via refinement of abstract problem specifications," Ph.D. dissertation, University of St Andrews, 2014.

[35] C. Borgelt, "Efficient implementations of apriori and eclat," in *FIMI'03: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations*, 2003.

[36] N. Een, "MiniSat: A SAT solver with conflict-clause minimization," in *Proc. SAT-05: 8th Int. Conf. on Theory and Applications of Satisfiability Testing*, 2005, pp. 502–518.

[37] C. Schulte, M. Lagerkvist, and G. Tack, "Gecode," *Software download and online material at the website: http://www.gecode.org*.