Itemset Mining as a Challenge Application for Answer Set Enumeration*

Matti Järvisalo

Department of Computer Science, University of Helsinki, Finland

Abstract. We present an initial exploration into the possibilities of applying current state-of-the-art answer set programming (ASP) tools—esp. conflict-driven answer set enumeration—for mining itemsets in 0-1 data. We evaluate a simple ASP-based approach experimentally and compare it to a recently proposed framework exploiting constraint programming (CP) solvers for itemset mining.

1 Introduction

Answer set programming (ASP) has become a viable approach to solving various hard combinatorial problems. This success is based on the combination of an expressive modeling language allowing high-level declarations and optimized black-box solver technology following the success story of Boolean satisfiability (SAT) solving.

A somewhat specific feature of typical answer set solvers is support for enumerating all solutions (answer sets) of answer set programs. In this work we study an exciting novel application domain for answer set enumeration, namely, the data mining task of finding all *frequent itemset* from 0-1 data [1]. We show that itemset mining problems allow for simple and natural encodings as ASP with the help of the rich modeling language. Notably, typical itemset mining algorithm are somewhat problem specific, varying on the constraints imposed on itemsets. Surprisingly, constraint satisfaction techniques have only very recently been applied to itemset mining tasks [2,3]. In addition to the availability of black-box constraint solvers such as answer set enumerators, the additional benefit of constraint solvers is that the modeling languages enable solving novel itemset mining tasks by combining different itemset constraints in a natural way without having to devise new solving algorithms for specific mining tasks.

In this short paper, focusing on the standard [1] and maximal [4] frequent itemset mining problems, we evaluate the effectiveness of answer set enumeration as an itemset mining tool using a recent conflict-driven answer set enumeration algorithm [5], and compare this ASP approach to a recent approach based on generic constraint programming (CP) [2,3]. The results show that, even with simple encodings, ASP can be a realistic approach to itemset mining, and, on the other hand, that itemset mining is a well-motivated benchmark domain for answer set enumeration (adding to the currently relative few realistic applications of answer set enumeration).

* Research financially supported by Academy of Finland under grant 132812.

J. Delgrande and W. Faber (Eds.): LPNMR 2011, LNAI 6645, pp. 304–310, 2011.

2 Itemset Mining

Assume a set $\mathcal{I} = \{1, ..., m\}$ of *items* and a set $\mathcal{T} = \{1, ..., n\}$ of *transactions*. Intuitively, a transaction $t \in \mathcal{T}$ consists of a subset of items from $\mathcal{I}$. An itemset database $\mathcal{D} \in \{0,1\}^{n \times m}$ is a binary matrix of size $n \times m$ that represents a set of transactions. Each row $\mathcal{D}_t$ of $\mathcal{D}$ represents a transaction t that consists of the set of items $\{i \in \mathcal{I} \mid \mathcal{D}_{ti} = 1\}$, where $\mathcal{D}_{ti}$ denotes the value on the ith column and tth row of $\mathcal{D}$.

The subsets of $\mathcal{I}$ are called *itemsets*. In itemset mining we are interested in finding itemsets that satisfy pre-defined constraints relative to an itemset database $\mathcal{D}$. Let $\varphi : 2^{\mathcal{I}} \rightarrow 2^{\mathcal{T}}$ be a function that maps an itemset $I \subseteq \mathcal{I}$ to the set $T \subseteq \mathcal{T}$ of transaction in which all its items occur, that is, $\varphi(I) = \{t \in T \mid \forall i \in I : \mathcal{D}_{ti} = 1\}$. The dual of φ is the function $\psi : 2^{\mathcal{T}} \rightarrow 2^{\mathcal{I}}$ that maps a set of transactions $T \subseteq \mathcal{T}$ to the set of all items from I included in all transactions in T, that is, $\psi(T) = \{i \in I \mid \forall t \in T : \mathcal{D}_{ti} = 1\}$.

Standard Frequent Itemsets. Assume a transaction database $\mathcal{D}$ over the sets $\mathcal{T}$ of transactions and $\mathcal{I}$ of items, and additionally a *frequency threshold* $\theta \in \{0, \ldots, |\mathcal{T}|\}$. Then the (traditional) frequent itemset problem [1] consists of finding the solution pairs (I, T), where $I \subseteq \mathcal{I}$ and $T \subseteq \mathcal{T}$, such that

$$T = \varphi(I) \tag{1}$$

$$|T| \geq \theta. \tag{2}$$

The first constraint requires that T must include all transactions in $\mathcal{D}$ that include all items in I. The second constraint requires that I is a frequent itemset, that is, the number of transactions in $\mathcal{D}$ in which all items in I occur must be at least θ. Notice that the second (*minimum frequency*) constraint is an anti-monotonic one: any subset of a frequent itemset is also a frequent itemset relative to a given threshold θ.

Various refinements of the traditional frequent itemset problem have been proposed. In addition to the traditional version, in this paper we consider the problem of finding *maximal* frequent itemsets [4], that is, frequent itemsets that are superset-maximal among the frequent itemsets of a given transaction database.

Maximal Frequent Itemsets. In addition to the constraints (1) and (2), the *maximality* constraint imposed in the maximal frequent itemset problem is

$$|\varphi(I')| < \theta \quad \forall I' \supset I, \tag{3}$$

that is, all supersets of a maximal frequent itemset are infrequent. Maximal frequent itemsets are a condensed representation for the set of frequent itemsets, constituting a border in the subset lattice of $\mathcal{I}$ between frequent and infrequent itemsets.

3 Itemsets as Answer Sets

We now consider two simple encodings of the standard and maximal frequent itemset problems as answer set programs. Due to the page limit we do not review details of the answer set semantics or the language used for expressing answer set programs. For more details on the input language, we refer the reader to the user's guide [6] of the Potassco bundle that includes the answer set enumerator we apply in the experiments.

```
1. item(I) :- db(_,I).
2. transaction(T) :- db(T,_).
3. { in_itemset(I) } :- item(I).
4. in_support(T) :- { conflict_at(T,I) : item(I) } 0, transaction(T).
5. conflict_at(T,I) :- not db(T,I), in_itemset(I), transaction(T).
6. :- { in_support(T) } N-1, threshold(N).
```

Fig. 1. The ASP(1) encoding of standard frequent itemset mining

```
1. item(I) :- db(_,I).
2. transaction(T) :- db(T,_).
3. { in_itemset(I) } :- item(I), N { in_support(T) : db(T,I) }, threshold(N).
4. in_support(T) :- { conflict_at(T,I) : item(I) } 0, transaction(T).
5. conflict_at(T,I) :- not db(T,I), in_itemset(I), transaction(T).
```

Fig. 2. The ASP(2) encoding of standard frequent itemset mining

We will intuitively explain the considered encoding referred to as ASP(1) (see Fig. 1) and ASP(2) (see Fig. 2). For both of the encodings, each answer set corresponds to a unique solution (I, T) of the itemset mining problem. Notice that there is an answer set for any dataset $\mathcal{D}$ and any threshold value θ, since by definition the empty set $\emptyset$ is always a frequent itemset. Although the encodings are quite similar, experiments show that the behavior of a state-of-the-art answer set enumerator varies notably depending of which encoding is used.

For presenting the transaction database $\mathcal{D}$, we use the predicate `db/2` and introduce the fact `db(t,i)` if and only if $\mathcal{D}_{ti} = 1$. The threshold θ is encoded using the predicate `threshold/1` by introducing the fact `threshold`(θ). The predicate `in_itemset/1` is true for an item i if and only if i is included in a frequent itemset I, encoding the most important part of a solution (I, T). The predicate `in_support/1` is true for a transaction t if and only if $t \in T$. Here the intuition is that, according to Eq. 1, each $t \in T$ has to *support* each $i \in I$ in the sense that t must include i (that is, $\mathcal{D}_{ti} = 1$). Additionally, we use the auxiliary predicates `item/1` (true for each item in $\mathcal{D}$), `transaction/1` (true for each transaction in $\mathcal{D}$), and `in_conflict/2`. The predicate `in_conflict/2`(t, i) is true for (t, i) if and only if transaction t does not support item i, that is, we have the *conflict* $\mathcal{D}_{ti} = 0$ and $i \in I$, violating Eq. 1.

Standard Frequent Itemset Mining. First consider the case of standard frequent itemset mining. Lines 1-2 in ASP(1) and ASP(2) are the same, simply stating that if $\mathcal{D}_{ti} = 1$ for some t, then i is an item (line 1), and similarly for transactions (line 2). The fact that a transaction t supports an itemset is also encoded in the same fashion in ASP(1) and ASP(2) on lines 4-5. Transaction t is in the support only if there is no conflict between t and the items in the itemset, that is, the number of true `conflict_at(`t, i`)`'s is zero (line 4, using a *cardinality constraint*). The `conflict_at/2` predicate is then defined on line 5: there is a conflict if $\mathcal{D}_{ti} = 0$ where i is in the frequent itemset.

The ASP(1) and ASP(2) encodings differ in how inclusion of items in the frequent itemset is represented. In ASP(1), on line 3 we "guess" for each item whether it is in the frequent itemset (`{ in_itemset(`i`) }` is the so called *choice* atom that is true

regardless of whether `in_itemset(`i`)` is true). Given any choice of included items, the *integrity constraint* of line 6 requires that the number of transactions supporting the chosen itemset cannot be less than the frequency threshold, in accordance with the minimum frequency constraint (Eq. 2).

In ASP(2), we apply a more "direct" way of encoding inclusion of items in frequent itemsets (line 3): there is the choice of including an item if the particular item has enough supporting transactions (that is, at least as many as required by the threshold θ).

Maximal Frequent Itemset Mining. Based on the encodings for standard frequent itemset mining, including the additional maximality criterion for frequent itemsets requires only a small modification to both ASP(1) and ASP(2). Namely, for ASP(1) we *add* the rule `in_itemset(I) :- item(I), N { in_support(T) : db(T,I) }, threshold(N).` enforcing that any item that has sufficient support for inclusion in the frequent itemset has to be included. In contrast, for ASP(2) we *replace* the rule on line 3 with this same rule, in essence removing the *choice* from the original rule.

4 Experiments

Here we report on preliminary experiments addressing the efficiency of a state-of-the-art answer set enumerator on the ASP(1) and ASP(2) encodings of real-world datasets. As the answer set enumerator, we use the conflict-driven solver Clingo [5] (version 3.0.3, based on the Clasp ASP solver version 1.3.5)[1] with default settings. We also compare the performance of Clingo on ASP(1) and ASP(2) to that of FIM_CP[2] version 2.1 (using Gecode `http://www.gecode.org/` version 3.2.2), which is a recently proposed tool for itemset mining based on constraint programming [2,3]. The experiments were conducted under Ubuntu Linux on a 3.16-GHz Intel CORE 2 Duo E8500 CPU using a single core and 4-GB RAM. As benchmarks we used the preprocessed UCI datasets available at `http://dtai.cs.kuleuven.be/CP4IM/datasets/`, as used in evaluating FIM_CP [2,3]. Key properties of representative datasets, as supplied at this website, are shown in Table 1. We ran each solver for threshold θ values $0.95, 0.90, \ldots, 0.10, 0.08, \ldots, 0.02, 0.01$ times $|\mathcal{I}|$ for each dataset $\mathcal{D}$ until we observed the first timeout for a particular solver.

Table 1. Properties of the representative datasets

Dataset $\mathcal{D}$	transactions	items	density (%)	itemsets at $\theta = 0.1 \cdot \lvert\mathcal{I}\rvert$		grounding time (s)	
				standard	maximal	ASP(1)	ASP(2)
anneal	812	93	45	$>$ 147 000 000	15 977	< 0.3	< 0.3
australian-credit	653	125	41	$>$ 165 000 000	2 580 684	< 0.3	< 0.3
lymph	148	68	40	9 967 402	5191	< 0.1	< 0.1
mushroom	8124	119	18	155 734	453	< 3.4	< 2.5
splice-1	3190	267	21	1606	988	< 3.8	< 2.8

[1] `http://potassco.sourceforge.net/`. The options `-n 0 -q` were used for computing all solutions and suppressing printing of solutions.

[2] `http://dtai.cs.kuleuven.be/CP4IM/`. The option `-output none` was used for suppressing printing of solutions.

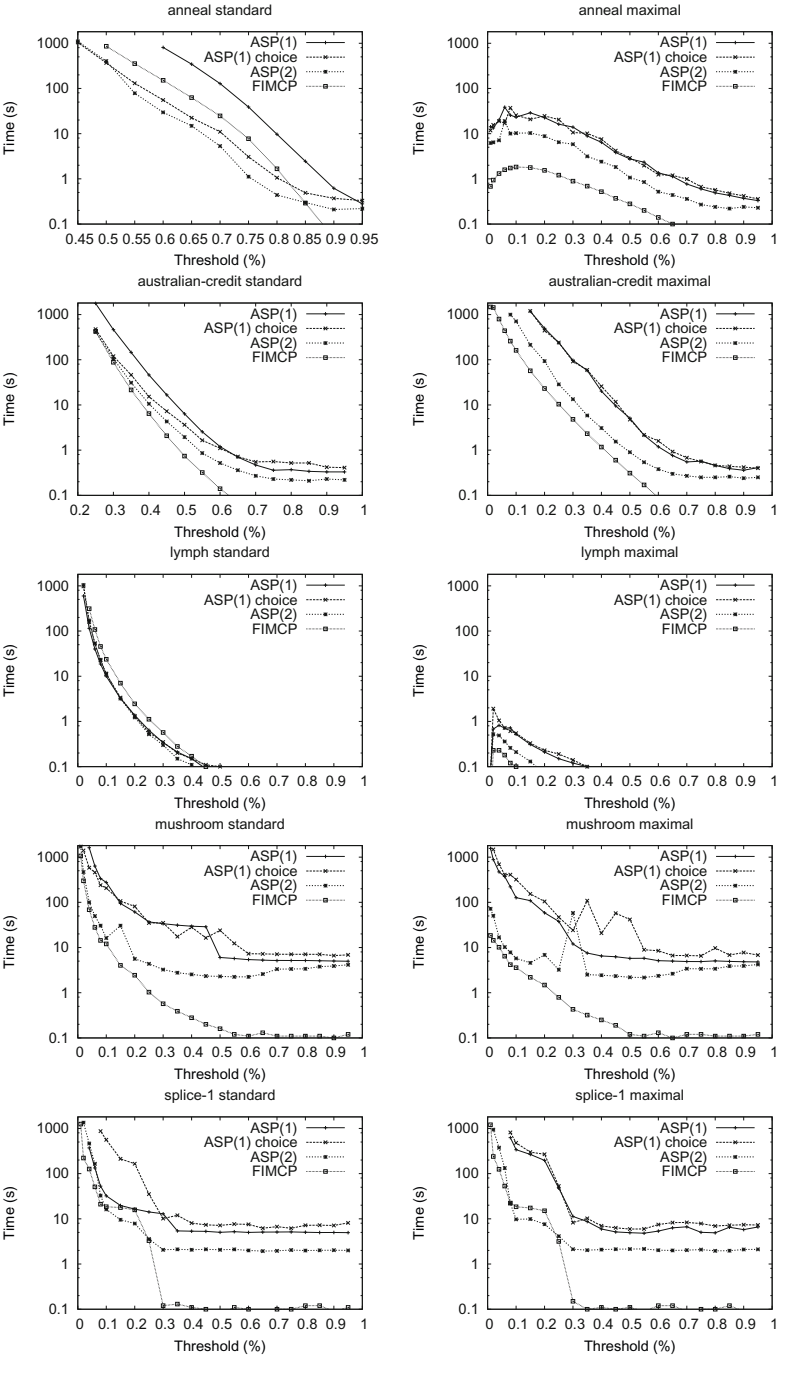

Fig. 3. Comparison of the CP and ASP approaches to standard and maximal frequent itemset mining on representative datasets

Results for a representative set of benchmarks are shown in Fig. 3, with observed upper bounds on the times used for grounding shown in Table 1. Grounding time is included in the plots. For the standard frequent itemset problem (left column), we observe that the ASP(2) encoding is almost always better than ASP(1). For the most dense dataset *anneal* (recall Table 1 – here density is defined as the percentage of 1's in $\mathcal{D}$) we observe that ASP(2) is the most effective one, being multiple times more effective than the FIM_CP approach. Also for the other two relatively dense datasets, ASP(2) is either slightly better than (on *lymph*) or approaches the performance (on *australian-credit*) of FIM_CP. This is an intriguing observation, since dense datasets can be considered harder to mine because of the large number of candidate itemsets. For the remaining two datasets, we observe that the performance of ASP(2) approaches that of FIM_CP, even being more effective at low threshold values on *splice-1*.

For the maximal itemset problem (right column) we observe that FIM_CP is the most efficient one, with the exception that for the *splice-1* dataset, the ASP(2) encoding dominates at the more difficult threshold values $\leq 0.20 \cdot |\mathcal{I}|$.

We also conducted a preliminary experiment on the effect of *decomposing* the cardinality and choice constructs in ASP(1) and ASP(2) using the build-in decompositions of Clingo. This is motivated by evidence of varied applications of constraint satisfaction tools in which decomposing complex constraints into lower level entities has resulted in improved performance. In this case, decomposing cardinalities seemed to generally degrade performance. However, decomposing only the choice constructs (using `--trans-ext=choice` in Clingo) in the standard frequent itemset encodings gave interesting results; see "ASP(1) choice" in Fig. 3. Namely, the performance of ASP(1) on *anneal* became even better than that of FIM_CP, but degraded further on *splice-1*. For ASP(2) we observed no notable differences.

Finally, we noticed that Smodels (with and without lookahead) is very ineffective on these problems compared to Clasp, and hence we excluded the Smodels data from the plots for clarity. However, in-depth experiments with other solution enumerating solvers (including, e.g., DLV) remains as future work, in addition to experimenting with different search heuristics and other search space traversal options offered by Clasp.

5 Conclusions

We propose itemset mining as a novel application and benchmark domain for answer set enumeration. The behavior of two simple ASP encodings varies depending on whether maximality of itemsets is required; the behavior of the "better" encoding can exceed that of a recent CP-based approach. We also observed that even small changes in the encoding—including decompositions—can reflect in notable performance differences when enumerating all solutions. This motivates further work on more effective encodings and on the interplay between answer set enumeration search techniques and modelling, with the possibility of optimizing solver heuristics towards data mining tasks. Additional current work includes finding dataset properties that imply good performance of the ASP approach, and encodings of other data mining tasks as ASP.

References

1. Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., Verkamo, A.I.: Fast discovery of association rules. In: Advances in Knowledge Discovery and Data Mining, pp. 307–328. AAAI Press, Menlo Park (1996)
2. De Raedt, L., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: Proc. KDD, pp. 204–212. ACM, New York (2008)
3. De Raedt, L., Guns, T., Nijssen, S.: Constraint programming for data mining and machine learning. In: Proc. AAAI. AAAI Press, Menlo Park (2010)
4. Burdick, D., Calimlim, M., Flannick, J., Gehrke, J., Yiu, T.: MAFIA: A maximal frequent itemset algorithm. IEEE Trans. Knowl. Data Eng. 17(11), 1490–1504 (2005)
5. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set enumeration. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 136–148. Springer, Heidelberg (2007)
6. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: A user's guide to `gringo`, `clasp`, `clingo`, and `iclingo` (2008), `http://potassco.sourceforge.net/`