Knowledge Compilation for Itemset Mining[1]

Hadrien Cambazard and **Tarik Hadzic** and **Barry O'Sullivan** [2]

Abstract. We present a novel approach to itemset mining whereby the set of all itemsets are compiled into a compact form, closely related to binary decision diagrams. While there were previous attempts to utilize decision diagrams for storing the set of frequent itemsets this is the first approach that does not rely on backtrack search to generate such a set. Our empirical evaluation demonstrates that our approach is complementary to current approaches.

1 INTRODUCTION

Mining frequent itemsets is a central problem in data mining. Given a database defined as a set of *transactions*, each of which is defined by a set of *items* (see Figure 1), the task is to find all sets of items (patterns) that are considered *frequent*. An itemset is frequent if it occurs at least a specified number of times in the database. We propose a novel approach that casts itemset mining as a *knowledge compilation* task. Our compilation-based approach supports a number of powerful queries that involve reasoning about the set of all frequent itemsets in time polynomial in the size of the compiled representation (which might be exponentially large in the worst case). Counting frequent items is one such query. The *technical advance* is to compile the set of all itemsets into a special form of *binary decision diagram* (BDD) [1], augmented with *counting variables*, so that all itemsets of the same frequency are represented by BDD paths that end in the same counting node. This representation is similar to the one used in [5] to compress the database as opposed to the itemsets. It was later abandoned by the same authors in [3]. The *novelty* of our approach is that it differs fundamentally from earlier attempts to incorporate knowledge compilation into itemset mining since it does not rely on search at all, but only proceeds by manipulation of BDDs. Therefore, any BDD manipulation package, such as [4], can be easily used with our approach as an itemset mining system.

2 FORMALISM

Itemset Mining. Let $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}$ be a set of n items and $\mathcal{D}$ be a database of m transactions denoted $\{T_1, T_2, \ldots, T_m\}$, in which each transaction T_i is defined as a subset of items, $T_i \subseteq \mathcal{I}$. A set X of items is called an *itemset* (or a pattern) and the *transaction set* of X in the database, denoted $T(X)$, is the subset of transactions in which the itemset occurs as a subset. The *support* of X, denoted $\sigma(X)$, is the number of such transactions $|T(X)|$. An itemset is said to be frequent with respect to a given threshold θ if it occurs in at least θ transactions, i.e. $\sigma(X) \geq \theta$.

[1] Supported by Science Foundation Ireland (Grant Numbers 05/IN/I886 and 05/IN.1/I886 TIDA 09).

[2] Cork Constraint Computation Centre, University College Cork, Ireland. {h.cambazard,t.hadzic,b.osullivan}@4c.ucc.ie

An example transaction database containing three transactions defined in terms of three items, $\{I_1, I_2, I_3\}$, is shown in Figure 1. The transactions are $T_1 = \{I_1, I_2, I_3\}$, $T_2 = \{I_1, I_3\}$, $T_3 = \{I_2, I_3\}$ (left). A frequency table, giving for each itemset X its transaction set and its support is shown on the right.

T_1 :	I_1, I_2, I_3
T_2 :	I_1, I_3
T_3 :	I_2, I_3

X	$T(X)$	$\sigma(X)$
$\emptyset$	$\{T_1, T_2, T_3\}$	3
$\{I_1\}$	$\{T_1, T_2\}$	2
$\{I_2\}$	$\{T_1, T_3\}$	2
$\{I_3\}$	$\{T_1, T_2, T_3\}$	3
$\{I_1, I_2\}$	$\{T_1\}$	1
$\{I_1, I_3\}$	$\{T_1, T_2\}$	2
$\{I_2, I_3\}$	$\{T_1, T_3\}$	2
$\{I_1, I_2, I_3\}$	$\{T_1\}$	1

Figure 1. An example transaction database.

Binary Decision Diagrams. *Binary decision diagrams* (BDDs) [1] are rooted directed acyclic graphs $G = (V, E)$ that represent the set of assignments over a set of Boolean variables, $\{x_1, \ldots, x_n\}$. A BDD has a root node, denoted **r**, and two terminal nodes **1** and **0** indicating true and false, respectively. Every nonterminal node is labeled with one of the Boolean variables x_i and has two outgoing edges, which encode assignments $x_i = 0$ and $x_i = 1$. BDDs are normally *ordered* unless stated otherwise – variables labeling nodes in a path from the root to the terminal are always in the same order, e.g. $x_1 \prec \ldots \prec x_n$. If an edge "skips" a variable x_i, then both $x_i = 0$ and $x_i = 1$ are supported. This encoding associates every path from the root to terminal with a partial assignment.

3 THE COMPILATION APPROACH

Our basic approach is to compile the set of all itemsets into a compact representation so that for each threshold θ we can distinguish the set of items whose support is at least θ. We introduce n Boolean variables $x_1, \ldots, x_n$, one for each item. Assignment $x_j = 1$ indicates that the item I_j is in the itemset. Hence, an assignment to $x_1, \ldots, x_n$ represents an itemset $I(x_1, \ldots, x_n) = \{I_j \mid x_j = 1, j = 1, \ldots, n\}$.

Figure 2. MTBDD for the support function σ. Each path corresponds to a set of itemsets and its terminal node represents the corresponding support.

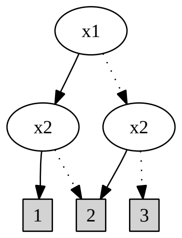

Table 1. Comparing the general compilation scheme “BDD all θ” with a restricted one, “BDD 1%” and *LCM count*

Instance			BDD all θ			BDD 1%				LCM count 1%
	n	**m**	$\mathbf{m_{succ}}$	**BDD size**	**Time**	θ	$\mathbf{m_{succ}}$	**BDD size**	**Time**	**Time**
Zoo-1	36	101	101	6750	0m 0.06s	1	101	1386	0m 0.28s	0m 0.1s
Vote	48	435	435	258158	0m 24s	4	435	39992	0m 11.5s	0m 0.3s
Tic-tac-toe	27	958	958	30863	0m 12s	10	958	5505	0m 14.3s	0m 0.1s
*Splice-1	287	3190	161	19247569	16m 20s	32	161	19246773	23m 33s	0m 19.8s
Soybean	50	630	630	50451	0m 8s	6	630	9694	0m 6.6s	0m 0.1s
Primary-tumor	31	336	336	46569	0m 2s	3	336	5934	0m 0.9s	0m 0.1s
Mushroom	119	8124	8124	136413	8m 5s	81	8124	23947	8m 48.6s	0m 0.3s
*Kr-vs-kp	73	3196	1321	15856014	109m 7s	32	*3196*	19017648	251m 28s	12m 55.6s
*Hypothyroid	88	3247	2121	13516004	146m 15s	32	*3247*	1052365	57m 38.1s	18m 3.2s
Hepatitis	68	137	137	1583304	0m 44s	1	137	184727	0m 6.7s	0m 12.2s
*Heart-cleveland	95	296	229	21279153	23m 59s	3	*296*	1405948	29m 35s	9m 24.1s
*German-credit	112	1000	505	19260913	54m 12s	10	646	19539742	108m 36s	3m 3.3s
*Australian-credit	125	653	292	na	20m 33s	7	380	20073775	45m 0s	45m 19.7s
Audiology	148	216	216	3185830	0m 44s	2	216	26265	0m 1.97s	> 10 hours
Anneal	93	812	812	931935	3m 40s	8	812	23164	0m 14.11s	0m 4.9s

We also introduce m Boolean variables $t_1, \ldots, t_m$, one for each transaction. An assignment to $t_1, \ldots, t_m$ represents a set of transactions $T(t_1, \ldots, t_m) = \{T_j \mid t_j = 1, j = 1, \ldots, m\}$. Finally, we introduce m constraints $C_1, \ldots, C_m$, one for each transaction T_i:

$$C_i : \quad t_i \Leftrightarrow \bigwedge_{I_j \notin T_i} \neg x_j, \;\; i = 1, \ldots, m$$

where t_i is a shorthand for $t_i = 1$ and $\neg x_j$ means $x_j = 0$. The constraint C_i ensures that a transaction T_i is in a transaction set exactly when none of the items outside the transaction ($I_j \notin T_i$) are in the itemset. It is not hard to see that any assignment to $(x_1, \ldots, x_n, t_1, \ldots, t_m)$ that satisfies all constraints $C_1, \ldots, C_m$, denoted as $(x_1, \ldots, x_n, t_1, \ldots, t_m) \models C_1 \wedge \ldots \wedge C_m$, represents an itemset $I(x_1, \ldots, x_n)$ and its corresponding transaction set $T(t_1, \ldots, t_m)$.

Our contribution is an approach that, starting with the constraint model above, constructs a BDD for the support function σ, such that $\sigma(x_1, \ldots, x_n) = |\{(t_1, \ldots, t_m) \mid (x_1, \ldots, x_n, t_1, \ldots, t_m) \models C_1 \wedge \ldots \wedge C_m\}|$. Such a BDD effectively encodes a *multi-terminal binary decision diagram* (MTBDD), where each terminal node corresponds to an itemset count. An MTBDD for our example is shown in Figure 2. We construct it using only standard BDD operations, such as conjunction and existential quantification, which are normally supported by BDD packages such as [4]. More details about the approach can be found in the extended version of this paper.[3]

4 EXPERIMENTS

We evaluated our approach on the 15 datamining instances used in [2].[4] The experiments ran as a single thread on a Dual Quad Core Xeon CPU, 2.66GHz with 12MB of L2 cache per processor and 16GB of RAM overall, running Linux 2.6.25 x64. The baseline used for comparison is the state of the art itemsets miner LCM [6]. We will refer to *LCM count* as the version of LCM that only counts the number of itemsets without storing them.

Table 1 reports the results. The number of items n and number of transactions m of each instance are indicated. The column “BDD all θ” reports results for computing count BDDs. Column $\mathbf{m_{succ}}$ shows the number of transactions that were conjoined successfully. Column **BDD size** reports the number of BDD nodes in the resulting BDD (in case of unsuccessful termination, this was the size of BDD at the last successful iteration). The second compilation scheme, “BDD 1%”, attempts to build a BDD restricted to a given threshold of 1% by merging all counting nodes for frequencies greater than θ, and pruning itemsets with count 0 in the final BDD. Finally the last column reports the time needed by *LCM count* for the same threshold of 1%.

The results are positive: 9 out of 15 instances can be solved for all thresholds at once by the general compilation approach. The method fails for memory reasons on 6 instances (indicated with a *) of which 3 can be handled when restricting the compilation to a threshold of 1%. *LCM count* is only comparable to the restricted scheme “BDD 1%” and proves to be much faster in general. However it fails on *Audiology* which appears to be easy even for the complete compilation (44s). It highlights that the compilation method can complement traditional approaches based on search and can handle instances that are at the moment out of reach of backtracking based methods.

5 CONCLUSIONS

Pattern discovery is typically an iterative task where queries are refined depending on the answers to previous queries. It is therefore important to develop techniques for incremental mining. The compilation approach presented in this paper addresses this issue by proposing a compact form to allow efficient queries regarding any threshold of the frequency.

REFERENCES

[1] Randal E. Bryant, ‘Graph-Based Algorithms for Boolean Function Manipulation’, *IEEE Transactions on Computers*, **35**, 677–691, (1986).

[2] Luc De Raedt, Tias Guns, and Siegfried Nijssen, ‘Constraint programming for itemset mining’, in *KDD ’08: Proceeding of ACM SIGKDD*, pp. 204–212, New York, NY, USA, (2008). ACM.

[3] Shin ichi Minato, Takeaki Uno, and Hiroki Arimura, ‘Lcm over zbdds: Fast generation of very large-scale frequent itemsets using a compact graph-based representation’, in *PAKDD*, pp. 234–246, (2008).

[4] J. Lind-Nielsen, ‘BuDDy - A Binary Decision Diagram Package’. `http://sourceforge.net/projects/buddy`, 2001.

[5] Shin-Ichi Minato and Hiroki Arimura, ‘Frequent pattern mining and knowledge indexing based on zero-suppressed bdds’, 152–169, (2007).

[6] Takeaki Uno, Masashi Kiyomi, and Hiroki Arimura, ‘Lcm ver.3: collaboration of array, bitmap and prefix tree for frequent itemset mining’, in *OSDM ’05: Proceedings of the 1st international workshop on open source data mining*, pp. 77–86, New York, NY, USA, (2005). ACM.

[3] `http://4c.ucc.ie/~thadzic/publications/Itemset.pdf`
[4] `http://www.cs.kuleuven.ac.be/~dtai/CP4IM/datasets/`