On Maximal Frequent Itemsets Enumeration

Said Jabbour[1(✉)], Fatima Zahra Mana[1,2], and Lakhdar Sais[1]

[1] CRIL-CNRS, Université d'Artois, 62307 Lens Cedex, France
{jabbour,sais}@cril.fr, fatimana93@gmail.com
[2] INPT, Institut national des postes et télécommunications, Rabat, Morocco

Abstract. Enumerating interesting patterns from data is an important data mining task. Among the set of possible relevant patterns, maximal frequent patterns is a well known condensed representation that limits at least to some extent the size of the output. Recently, a new declarative mining framework based on constraint programming (CP and satisfiability (SAT) has been designed to deal with several pattern mining tasks. For instance, the itemset mining problem has been modeled as a constraint network/propositional formula whose models correspond to the pattern to be mined. In this framework, closeness, maximality and frequency properties can be handled by additional constraints/formulas. In this paper, we propose a new propositional satisfiability based approach for mining maximal frequent itemsets that extends the one proposed in [13]. We show that instead of adding constraints to the initial SAT based itemset mining encoding, the maximal itemsets, can be obtained by performing clause learning during search. Our approach leads to a more compact encoding. Experimental results on several datasets, show the feasibility of our approach.

1 Introduction

Mining frequent itemsets in datasets is a fundamental problem in the data mining field since it enables several mining tasks such as discovering association rules, data correlations, sequential patterns, etc. The problem of finding frequent itemsets and its corresponding association rules was originally proposed by Agrawal in [2]. Several algorithms have been proposed to deal with this enumeration problem. One can cite the levelwise and the pattern-growth like approaches. The first one is based on the generate-and-test framework [2], while the second is based on the divide-and-conquer framework [9] (see also H-Mine [18], LCM [22] and [21] for a survey).

Usually, the number of the frequent patterns is known to be large. To reduce the size of the output, condensed representations, such as maximal and closed itemsets, have been introduced. Maximal Frequent Itemsets (MFI) is a subset of closed itemsets with longest size. Several methods have been proposed to discover the maximal frequent itemsets. Bayardo proposed a MaxMiner algorithm which extends the Apriori algorithm [15]. MaxMiner employs a breadth-first traversal of the search space to limit the database scanning. Furthermore,

© Springer International Publishing AG, part of Springer Nature 2018
A. Abraham et al. (Eds.): SoCPaR 2017, AISC 737, pp. 151–160, 2018.
https://doi.org/10.1007/978-3-319-76357-6_15

it uses a dynamic heuristic to increase the effectiveness of superset-frequency pruning. Several other enhancements have been suggested for mining the MFI. Pincer-Search algorithm combined the top-down and bottom-up [16] techniques to discover the maximal frequent itemset. Agarwal et al. implement a depth first search technique with bitmap representation (DepthProject) [1]. In which, column denotes the items and rows denotes the transactions. Like MaxMiner algorithm, they used dynamic reordering and look-ahead pruning. A projection mechanism is used to reduces the size of database. They efficiently find the support counts and give a superset of the MFI. D. Burdick et al. further extended DepthProject and named it as Mafia [4]. They used vertical bit-vector data format. Compression and projection on bitmaps are applied to increase the performance. Unlike DepthProject and MaxMiner pruning technique, Mafia used Parent Equivalence Pruning. Algorithm GenMax [8] is a backtrack search based algorithm. More specifically, it integrates numerous optimization techniques to prune the search space including progressive focusing that perform maximality checking and diffset propagation for fast support counting. To search MFI, SmartMiner [24] records at each step tail information to guide the search for new MFI.

Recently, constraint programming and propositional satisfiability have been used for modeling several data mining tasks in a declarative way, and solving them using generic solving techniques [5,7,11,13,19]. Such approaches show that many task in data mining such as itemset, association rules, and sequence mining can be reduced into the enumeration of the models of a set of constraints/propositional formulas. In [20], the authors show that the generation of MFI can also be formulated as the enumeration of a set of models of a constraint network by adding a constraint to force the required models to be maximal.

In this paper, we propose to extend the SAT based approach for mining closed frequent itemsets proposed in [13], to generate the set of maximal ones. We show that instead of encoding maximality as a set of constraints, we can add clauses during search to cut non-maximal frequent itemsets.

2 Background

Let us first introduce the propositional satisfiability problem (SAT) and some necessary notations. We consider the conjunctive normal form (CNF) representation for the propositional formulas. A *CNF formula* Φ is a conjunction ($\wedge$) of clauses, where a *clause* is a disjunction ($\vee$) of literals. A *literal* is a positive (p) or negated ($\neg p$) propositional variable. The two literals p and $\neg p$ are called *complementary*. A CNF formula can also be seen as a set of clauses, and a clause as a set of literals. We denote by $Var(\Phi)$ the set of propositional variables occurring in Φ.

A *Boolean interpretation* $\mathcal{B}$ of a propositional formula Φ is a function which associates a value $\mathcal{B}(p) \in \{0, 1\}$ (0 corresponds to $false$ and 1 to $true$) to the propositional variables $p \in Var(\Phi)$. It is extended to CNF formulas as usual. A *model* of a formula Φ is a Boolean interpretation $\mathcal{B}$ that satisfies the formula, i.e., $\mathcal{B}(\Phi) = 1$. We note $\mathcal{M}(\Phi)$ the set of models of Φ. *SAT problem* consists in deciding if a given formula admits a model or not.

Algorithm 1. DPLL Based Enumeration solver

```
Input: $\Phi$: a CNF formula
Output: $\mathcal{M}$: all the models of $\Phi$
$\mathcal{I} = \emptyset$, ;                              /* Current interpretation */
$\mathcal{M} = \emptyset$ ;                                      /* Set of models */
$dl = 0$ ;                                                     /* decision level */
while (true) do
    $conflictClause$ = unitPropagation($\Phi$, $\mathcal{I}$);
    if ($conflictClause$!=$null$) then
        if ($dl == 0$) then return M;
        $\ell = lastDecision$;
        $backtrack(dl - 1)$;
        $dl = dl - 1$;
        $\mathcal{I} = \mathcal{I} \cup \{\neg\ell\}$;
    else
        if ($\mathcal{I} \models \Phi$) then
            $\mathcal{M} = \mathcal{M} \cup \{\mathcal{I}\}$;
            $\ell = lastDecision$;
            backtrack(dl-1);
            $dl = dl - 1$;
            $\mathcal{I} = \mathcal{I} \cup \{\neg\ell\}$;
        else
            $\ell$ = selectDecisionVariable($\Phi$);
            $dl = dl + 1$;
            $\mathcal{I} = \mathcal{I} \cup \{\ell\}$;
        end
    end
end
```

Let us informally describe the most important components of modern SAT solvers, usually called CDCL (Conflict Driven Clause Learning) solvers. They are based on a reincarnation of the historical Davis, Putnam, Logemann and Loveland procedure, commonly called DPLL [6]. It performs a backtrack search; selecting at each level of the search tree, a decision variable which is set to a Boolean value. This assignment is followed by an inference step that deduces and propagates some forced unit literal assignments. This is recorded in the implication graph, a central data-structure, which encodes the decision literals together with there implications. This branching process is repeated until finding a model or a conflict. In the first case, the formula is answered satisfiable, and the model is reported, whereas in the second case, a conflict clause (called learnt clause) is generated by resolution following a bottom-up traversal of the implication graph [17,23]. The learning or conflict analysis process stops when a conflict clause containing only one literal from the current decision level is generated. Such a conflict clause asserts that the unique literal with the current level (called asserting literal) is implied at a previous level, called assertion level, identified as the maximum level of the other literals of the clause. The solver backtracks to the assertion level and assigns that asserting literal to *true*. When an empty conflict clause is generated, the literal is implied at level 0, and the original formula can be reported unsatisfiable. In addition to this basic scheme, modern SAT solvers use other components such as activity based heuristics and restart policies. An extensive overview can be found in [3].

When tackling the enumeration of the models of a CNF formula, the clause learning components can be heavy when the set of models to learn is very large. In [10], the authors show that a simple DPLL algorithm can outperform CDCL based approach when dealing with the problem of enumerating models of propositional formulas. The efficiency of the proposed enumeration algorithm have been evaluated on several propositional formulas encoding itemsets mining problem [14].

Algorithm 1 depicts the general scheme of a DPLL-like procedure to enumerate the models of a formula. when a model is found (line 16), then the solver performs a simple backtracking to the precedent level to propagate the negation of the last decision (line 18). The enumeration stops when a conflict occurs in level 0 or if the model is found at level 0.

3 Frequent Itemset Mining

A database D comprises a set of transactions $\{T_1, T_2, \ldots, T_m\}$ and Ω a set of items. Each transaction has a unique transaction identifier (tid) and contains a set of items over Ω. A set of items is often called an itemset. Let I be an itemset and T a transaction. We will use the notation, $I \subseteq T$, to denote that I is a subset of the set of items that T contains. When the context is clear, we will often directly refer to a transaction as the set of items that it contains.

Classical Data mining problems are usually concerned with itemsets that frequently occur in a database of transactions. The number of occurrences of an itemset in a database is commonly referred to as the support of this itemset, formalized as follows.

Definition 1 (Support). *Let X be an itemset and D a database of transactions. The support of X in D, denoted $Supp(X)$, is the number of transactions of D in which X occurs as a subset. The frequency of X is defined as the ratio of $Supp(X)$ to $|D|$.*

Let $\mathcal{D}$ be a transaction database over Ω and λ a minimum support threshold. The *frequent itemset mining problem* consists in computing the following set:

$$FI(D, \lambda) = \{X \subseteq \Omega \mid Supp(X) \geqslant \lambda\}.$$

Mining of the complete set of frequent itemsets may lead to a huge number of itemsets. In order to face this issue, this problem can be limited on the extraction of closed itemsets. Thus, enumerating all closed itemsets allows us to reduce the size of the output.

Table 1. A transaction database D

tid	Itemset			
1	A	B	C	D
2	A	B	E	F
3	A	B	C	
4	A	C	D	F
5	G			
6	D			
7	D	G		

Definition 2 (Closed Frequent Itemset). *Let D be a transaction database (over Ω). An itemset X is a closed itemset if there exists no itemset X' such that (1) $X \subseteq X'$ and (2) $\forall T \in D$, $X \in T \rightarrow X' \in T$.*

Extracting all the elements of $FI(\mathcal{D}, \lambda)$ can be obtained from the closed itemsets by computing their subsets. We denote by $CFI(\mathcal{D}, \lambda)$ the subset of all closed itemsets in $FI(\mathcal{D}, \lambda)$.

For instance, consider the transaction database described in Table 1. The set of closed frequent itemsets with the minimal support threshold equal to 2 are: $CFI(\mathcal{D}, 2) = \{A, D, G, AB, AC, AF, ABC, ACD\}$.

If we consider subset inclusion as defining a partial order for itemsets, then we can introduce the notions of maximal frequent itemsets, as follows.

Definition 3 (Maximal Frequent Itemset). *Let X be an itemset of a database D. We say that X is a maximal itemset in D given a minimum threshold λ, if there exists no itemset Y such that $X \subset Y$ and Y is a frequent in D.*

The problem of mining maximal frequent itemsets MFI, is to enumerate all maximal frequent itemsets whose support is no less than a preset threshold.

$$MFI(D, \lambda) = \{X \subseteq \Omega \mid Supp(X) \geqslant \lambda \text{ and } X \text{ is maximal}\}.$$

4 Itemset Mining Based Constraints Encoding

Recent work tackle the problem of mining frequent itemsets by encoding this problem into constraints. In [13], the authors show that the generation of itemsets from transaction D can be encoded as the enumeration of the models of a CNF formula Φ i.e., there exists a one to one mapping between the models of the set of constraints and the set of frequent itemsets. Let us first review this approach. As said, the approach of [13] consists to introduce boolean variables p_a (resp. q_i) to represent each item $a \in \Omega$ (resp. each transaction T_i).

$$\bigwedge_{i=1}^{m} (\neg q_i \leftrightarrow \bigvee_{a \in \Omega \setminus T_i} p_a) \tag{1}$$

$$\sum_{i=1}^{m} q_i \geqslant \lambda \tag{2}$$

$$\bigwedge_{a \in \Omega} ((\bigvee_{a \notin T_i} q_i) \vee p_a) \tag{3}$$

The formula (1) allows to model the transaction database and then to catch the itemsets when an itemset appear in a transaction T_i ($q_i = 1$) if and only iff the variables not involved in T_i are set to false. The formula $(\neg q_i \leftrightarrow \bigvee_{a \in \Omega \setminus T_i} p_a)$ can be translated to the following CNF formula:

$$\bigwedge_{a \in \Omega \setminus T_i} (\neg q_i \vee \neg p_a) \wedge (q_i \vee \bigvee_{a \in \Omega \setminus T_i} p_a)$$

Formula (2) allows us to consider the itemsets having a support greater than or equal to λ. This encoding is defined as a 0/1 linear inequality, usually called cardinality constraint. Because of the presence of such constraint in several applications, many efficient CNF encodings have been proposed over the years. Mostly, such encodings try to derive the best compact representation while preserving the efficiency of constraint propagation (e.g. [12]).

Formula (3) capture the closure property. Intuitively, if the itemset is involved in all transactions containing an item a then a must be added to the candidate itemset. In other words, when in all the transactions where a does not appear, the candidate itemset is not included, we deduce that the candidate itemset appears only in transactions containing the item a. Consequently, to be closed, the item a must be added to the final itemset.

The advantage of this approach is its ability to be easy to modify in order to integrate others constraints. For instance, enumerating itemsets of size at most k, can be expressed by simply adding the linear constraint $\sum_{a \in \Omega} p_a \leqslant k$.

5 Enumerating Maximal Itemset

In this section we show how to enumerate maximal itemset using constraints. Let us recall that in [20], the authors provide a set of constraints that can be added to the CSP formula of closed frequent itemsets problem in order to generate the set of maximal itemset. Similarly, using satisfiability the maximality can be expressed as follow:

$$\bigwedge_{a \in \Omega} (\neg p_a \rightarrow \sum_{T_i \ \mid \ a \in T_i} q_i < \lambda) \tag{4}$$

Constraint (4) expesses that if a is not in the final maximal itemsets I, it means that the frequency of I in the transactions containing a is lower than λ. However, translating constraint (4) into clauses can lead to a large CNF formula. Another alternative is to manage theses constraints inside the solver. However their number remains a problem for the efficiency of SAT solvers. To avoid constraint addition, in this section we show a new approach that allows to add blocking clauses during search. For this purpose, we consider that our solver is a DPLL-like procedure. To illustrate our approach, assume that the solver assign with the truth value true the variables representing items. Let us note $\mathcal{B}$ as the model of the CNF formula encoding the frequent itemsets mining task and $P(\mathcal{B}) = \{a|\mathcal{B}(p_a) = 1\}$ the frequent itemset. Clearly the first found model $\mathcal{B}$ corresponds to a maximal frequent itemset $P(\mathcal{B})$. To avoid retrieving a model $\mathcal{B}'$ such that $P(\mathcal{B}') \subset P(\mathcal{B})$, one need to block all itemset $I \subset \mathcal{B}$. To do this, it is sufficient to add the blocking clause $c = (\bigvee_{a \in \Omega \setminus P(\mathcal{B})} p_a)$ to Φ. The solver can then backtrack and performs positive assignment of the variables representing Ω.

The main idea of our approach consists in adding blocking clauses each time a model is found. Let us note that such clauses comprise the literals representing items that are assigned to false under the current assignment. Consequently, such clauses are false before backtracking. In order to enumerate correctly the set of maximal itemsets, one need to take into account the levels of literals of c to backtrack at the adequate level.

Note that in general for real transaction databases, the missing items in each transaction T_i is larger than those of T_i i.e., $|T_i| \ll |\Omega \setminus T_i|$. As a consequence, each blocking clause c added to cut non maximal itemsets can be larger. Let us suppose that the current itemset appears in the transaction T_i. Clearly, c can be written as $c = (\bigvee\limits_{a \in T_i \setminus P(\mathcal{B})} p_a \vee \bigvee\limits_{a \in \Omega \setminus T_i} p_a)$. On the other hand, according to constraints (1), $\neg q_i = \bigvee\limits_{a \in \Omega \setminus T_i} p_a$. So, c can be rewritten as $c = (\bigvee\limits_{a \in T_i \setminus P(\mathcal{B})} p_a \vee \neg q_i)$. As formulated, the size of c can be considerably reduced. Furthermore, one can obtain the smallest clause c by choosing the smallest transaction T_i containing $P(\mathcal{B})$.

In the sequel, we present Algorithm 2 that can be integrated to Algorithm 1 to generate maximal itemsets based on the approach discussed below. More precisely, and as discussed before, when a model is found, a blocking clause c must be added to the formula. This clause is falsified at the current level. Furthermore, the literals of c can be assigned at different level of the search tree. Then, instead of performing a simple backtracking as shown in Algorithm 1, we have to analyze c in order to deduce the adequate backtracking level.

Algorithm 2. DPLL for Maximal Itemsets

```
1 $\mathcal{M} = \mathcal{M} \cup \{\mathcal{B}\}$;
2 $c \rightarrow \bigvee_{a \in \Omega \mid \neg p_a \in \mathcal{B}} p_a$;
3 $\Phi \leftarrow \Phi \cup c$;
4 $btl, \ell \leftarrow analyze(c)$;
5 backtrackUntil($btl$);
6 $dl = btl$;
7 $\mathcal{B} = \mathcal{B} \cup \{\ell\}$;
```

Example 1. Let us reconsider the set of transactions of Table 1 and assume that $\lambda = 2$. Suppose that the solver choose the following variable ordering p_A, p_E, p_F, p_G, p_B, p_D, and p_C. Then, the first assignment leading to a model is $\mathcal{B} = \{p_A, \neg p_E, \neg p_F, \neg p_G, p_B, \neg p_D, p_C\}$ such that p_A is assigned to level 1, p_B at level 2, and p_C at level 3. The added blocking clause is $c = (p_D \vee p_E \vee p_F \vee p_G)$. Then the solver must backtrack to the level 1 since c is falsified in level 2.

From the complexity point of view, as shown, each time a model is found a blocking clause is added to the formula. Consequently, the maximum number of added clauses is equal to the number of maximal itemsets. Let us recall that the number of maximal itemsets is often limited compared to closed ones.

6 Experimental Validation

We carried out an experimental evaluation to analyze the effect of adding blocking clauses and branching heuristics. To this end, we implemented a DPLL-like procedure, denoted `DPLL4MFI` as described in Algorithm 2. In this procedure, each time a model is found, we add a blocking clause (no-good) and perform a backtracking after analyzing the blocking clause. We considered a variety of datasets taken from the `FIMI`[1] and `CP4IM`[2] repositories. All the experiments were done on Intel Xeon quad-core machines with 32 GB of RAM running at 2.66 GHz. For each instance, we used a timeout of 15 min of CPU time.

In our experiments, we compare the performances of `DPLL4MFI` to the variant `DPLL4CFI` that enumerate the set of closed itemsets. Let us recall that the DPLL procedure utilize a static branching heuristic i.e., the variables corresponding to the infrequent items are assigned first. Note that the solver does not branch on q_i variables. In fact, by assigning the variables p_a, $a \in \Omega$, the variables q_i, $1 \leq i \leq m$ are propagated because the variables q_i depend on those of p_a as expressed by constraint (1).

In Table 2 we report some obtained results when considering some datas and some chosen values of the minimum support threshold. Unsurprisingly, the

Table 2. Maximal Itemsets for some datas

instance (#item, #trans)	min_supp	#CFI	#MFI	DPLL4CFI time(s)	DPLL4MFI time(s)
hepatitis (68, 137)	14	1827263	189205	2.21	29.35
lumph (68, 148)	10	46801	5191	0.08	0.11
primary_tumor (31, 336)	34	31024	2043	0.04	0.02
anneal (93, 812)	81	1224754	15977	2.7	0.47
german-credit (112, 1000)	100	2080152	232107	10.38	37.26
mushroom (119, 8124)	812	3287	453	1.95	1.72

[1] FIMI: http://fimi.ua.ac.be/data/.
[2] CP4IM: http://dtai.cs.kuleuven.be/CP4IM/datasets/.

number of maximal itemsets is often limited compared to closed ones. Also, the time needed to compute them is slightly greater than the one needed to generate the closed ones due to added blocking clauses.

7 Conclusion

In this paper, we consider the problem of enumerating the set of maximal frequent itemsets using propositional satisfiability. We show how to answer this question by utilizing a DPLL-like procedure for model enumeration combined to clause learning from models. Experimental results show that this approach is interesting, as it allows to avoid adding maximality constraints to the initial encoding.

As a future work, we plan to pursue our investigation in order to find the best heuristics to speed up the enumeration of the models. For example, it would be interesting to integrate some background knowledge in such heuristic design. Finally, clause learning, an important component for the efficiency of SAT solvers, admits several limitation in the context of model enumeration. An important issue, is to study how such important mechanism can be efficiently integrated when maximal itemset generation is considered.

References

1. Agarwal, R.C., Aggarwal, C.C., Prasad, V.V.V.: Depth first generation of long patterns. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2000, pp. 108–118 (2000)
2. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, SIGMOD 1993, pp. 207–216. ACM, New York (1993)
3. Biere, A., Heule, M.J.H., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. Frontiers in AI and Applications, vol. 185. IOS Press, Amsterdam (2009)
4. Burdick, D., Calimlim, M., Gehrke, J.: MAFIA: a maximal frequent itemset algorithm for transactional databases. In: ICDE, pp. 443–452 (2001)
5. Coquery, E., Jabbour, S., Saïs, L., Salhi, Y.: A sat-based approach for discovering frequent, closed and maximal patterns in a sequence. In: Proceedings of the 20th European Conference on Artificial Intelligence (ECAI 2012), pp. 258–263 (2012)
6. Davis, M., Logemann, G., Loveland, D.W.: A machine program for theorem-proving. Commun. ACM **5**(7), 394–397 (1962)
7. Gebser, M., Guyet, T., Quiniou, R., Romero, J., Schaub, T.: Knowledge-based sequence mining with ASP. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9–15 July 2016 (2016)
8. Gouda, K., Zaki, M.J.: Efficiently mining maximal frequent itemsets. In: Proceedings of the 2001 IEEE International Conference on Data Mining, San Jose, California, USA, 29 November–2 December 2001, pp. 163–170 (2001)
9. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. SIGMOD Rec. **29**, 1–12 (2000)

10. Jabbour, S., Lonlac, J., Sais, L., Salhi, Y.: Extending modern SAT solvers for models enumeration. In: Proceedings of the 15th IEEE International Conference on Information Reuse and Integration, IRI 2014, Redwood City, CA, USA, 13–15 August 2014, pp. 803–810 (2014)
11. Jabbour, S., Sais, L., Salhi, Y.: Boolean satisfiability for sequence mining. In: 22nd ACM International Conference on Information and Knowledge Management (CIKM 2013), pp. 649–658. ACM (2013)
12. Jabbour, S., Sais, L., Salhi, Y.: A pigeon-hole based encoding of cardinality constraints. TPLP **13**(4–5-Online-Suppl.) (2013)
13. Jabbour, S., Sais, L., Salhi, Y.: The top-k frequent closed itemset mining using top-k SAT problem. In: European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2003), pp. 403–418 (2013)
14. Jabbour, S., Sais, L., Salhi, Y.: On SAT models enumeration in itemset mining. CoRR abs/1506.02561 (2015)
15. Bayardo Jr., R.J.: Efficiently mining long patterns from databases. In: SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, Seattle, Washington, USA, 2–4 June 1998, pp. 85–93 (1998)
16. Lin, D.I., Kedem, Z.M.: Pincer-search: a new algorithm for discovering the maximum frequent set, pp. 103–119 (1998)
17. Marques-Silva, J.P., Sakallah, K.A.: GRASP - a new search algorithm for satisfiability. In: Proceedings of IEEE/ACM CAD, pp. 220–227 (1996)
18. Pei, J., Han, J., Lu, H., Nishio, S., Tang, S., Yang, D.: H-mine: hyper-structure mining of frequent patterns in large databases. In: Proceedings IEEE International Conference on Data Mining, ICDM 2001, pp. 441–448 (2001)
19. Raedt, L.D., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: ACM SIGKDD, pp. 204–212 (2008)
20. Raedt, L.D., Guns, T., Nijssen, S.: Constraint programming for itemset mining. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, 24–27 August 2008, pp. 204–212 (2008)
21. Tiwari, A., Gupta, R., Agrawal, D.: A survey on frequent pattern mining: current status and challenging issues. Inf. Technol. J. **9**, 1278–1293 (2010)
22. Uno, T., Kiyomi, M., Arimura, H.: LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets. In: FIMI 2004, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK, 1 November 2004 (2004)
23. Zhang, L., Madigan, C.F., Moskewicz, M.W., Malik, S.: Efficient conflict driven learning in Boolean satisfiability solver. In: IEEE/ACM CAD 2001, pp. 279–285 (2001)
24. Zou, Q., Chu, W.W., Lu, B.: SmartMiner: a depth first algorithm guided by tail information for mining maximal frequent itemsets. In: Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), Maebashi City, Japan, 9–12 December 2002, pp. 570–577 (2002)