Association Rules Mining: Application to Large-Scale Satisfiability

Habiba Drias, Youcef Djenouri
LRIA, USTHB

Abstract

Big data mining is a promising technology for industrial applications such as business intelligence and marketing. In this paper, we attempt to transfer this background to a scientific domain, which is problem solving. An enormous research investment has been done over the past years for solving NP-complete problems. The exact methods yield the optimal solution if it exists but they are not able to tackle problems of large instances because they would generate a combinatorial explosion and a timeout calculation. Approximate approaches are mostly stochastic and do not guarantee to find a solution even if one exists. For both alternatives, and in order to reach more closely any optimal solution, the search space should be explored in a judicious manner. If we consider the solutions collection as a data set, data mining techniques may help acquiring some knowledge on the landscape fitness function and then exploiting the achieved information in solving the problem. We are especially interested in locating those regions that are promising in the sense they contain solutions of good quality. We aim especially at developing a two-phase approach to address the satisfiability problem in order to cope with its complexity.

In the first step, a preprocessing of clauses is undertaken using association rules mining. The achieved associative clauses classes are then exploited to solve the instance. Extensive experiments are performed on BMC benchmarks and numerical results show the benefit of our proposal.

1. Introduction

Frequent pattern mining has become widely used in many areas such as marketing and DNA sequence analysis. It consists in finding regularities that occur frequently in data sets. Association rules are a means to represent frequent patterns and do not guarantee to find a solution even if one exists. For both alternatives, and in order to reach more closely any optimal solution, the search space should be explored in a judicious manner. If we consider the solutions collection as a data set, data mining techniques may help acquiring some knowledge on the landscape fitness function and then exploiting the achieved information in solving the problem. We are especially interested in locating those regions that are promising in the sense they contain solutions of good quality. We aim especially at developing a two-phase approach to address the satisfiability problem in order to cope with its complexity.

In the first step, a preprocessing of clauses is undertaken using association rules mining. The achieved associative clauses classes are then exploited to solve the instance. Extensive experiments are performed on BMC benchmarks and numerical results show the benefit of our proposal.

This paper is a first attempt to explore big data mining for a scientific application such as problem solving. Solving NP-complete problems and the satisfiability problem especially is one of the trickier issues studied for more than half a century. Tremendous efforts have been spent around the world to shed the light on these complex problems and significant and various results have been achieved. However, investments are still needed for acquiring a better knowledge on these hard problems leading for a better resolution. One possible research direction may consist in exploring in a judicious way the search space especially for large problem instances before launching the resolution process. The idea is to develop two steps for solving the problem, the first one consists in preprocessing the instance by extracting knowledge on the solutions space nature and the second is to solve the problem using the acquired knowledge. Our work considers data mining techniques and especially association rules mining to explore the search space for the satisfiability problem. The aim is to reduce the complexity of the problem by clustering clauses and hence variables and afterwards solving the clusters with a smaller number of variables.

From the technical part, association rules mining have been commonly used in relational databases where data are represented in a table. An entry of the table represents a transaction and the columns its attributes. Several algorithms are proposed in the literature to address the issue. Three major methods have dominated associated rules mining: Apriori [2], FPgrowth [8] and Charm [13]. Apriori can be seen as a horizontal search method whereas Charm is another vertical version. Apriori has the disadvantage to be slow as it crosses the database at each iteration of the algorithm. FPgrowth aimed at palliating this issue by using a complex data structure namely the FPTree.

For our concern, we focus on exploiting association rules mining for preprocessing clauses of SAT instances. For this aim, the clauses represent the transactions and the variables the attributes. The variables are grouped into clusters that are solved using a SAT solver. Methods other than the classical association rules mining are proposed to cope with the scalability dimension. They are based on hybrid meta-heuristics and more precisely on a bio-inspired approach and a taboo search.

The rest of the paper is organized as follows. In section 2, the problem SAT is recalled and the related works are presented in section 3. The proposal and its implementation are exposed and discussed in sections 4, 5 and 6. The conducted experiments and the obtained results are exhibited in section 7. Finally, in the last section, we conclude our study and give some perspectives.

2. The Satisfiability Problem

Satisfiability (SAT) arouses more interest from the communities of artificial intelligence and computational complexity. Indeed, all eyes turn to this problem in recent years thinking that solving SAT could make significant progress in these two major disciplines. Of course, these advancements will generate a significant impact in the field of computer science in general.

2.1 Definitions

Given a set of Boolean variables V = {v1, v2, ..., vn} and an assignment function t: V → {T, F} that associates a truth value to each variable vi, the general notions that define SAT are:

Tapez une équation ici.
- a literal is a variable that appears with or without the negation operator.
- a clause is a disjunction of literals.
- an instance of SAT is a conjunction of clauses.
- the problem SAT is defined by the following (instance, question) pair

\[
\begin{align*}
\text{Instance:} & \text{ m clauses over n variables} \\
\text{Question:} & \text{ is there an instantiation of variables for which all the clauses are true?}
\end{align*}
\]

A clause is True (T) if and only if it includes at least one positive literal with the value T or one negative literal with the value False (F).

Example:

Let consider the following set of variables \( V = \{v_1, v_2, v_3, v_4\} \) and the following set of clauses \( C = \{C_1, C_2, C_3\} \) defined as:

- \( C_1 = v_1, v_2, \neg v_4 \)
- \( C_2 = v_2, \neg v_3, v_4 \)
- \( C_3 = \neg v_1, \neg v_2, v_3 \)

The \( \neg \) sign denotes the negation operator and \( \lor \), the disjunctive operator. One possible solution is the instantiation \( \{T, T, T, T\} \) respectively assigned to \( \{v_1, v_2, v_3, v_4\} \).

2.2 SAT Solvers

There are several approaches for solving SAT and a large number of SAT solvers have been developed since the 60s. They are used in the academic world as well as in the industry to solve complex problems. The first designed solvers are called "complete" because they are able to compute the optimal solution if it exists or prove that there is none. The most known SAT solvers are:

The Davis-Putnam Procedure (DPP)

DPP [4] is one of the complete algorithm that has been widely used for solving SAT. The idea of the algorithm is the construction of a binary tree using recursion and backtracking. By assigning values to variables and then eliminating these variables from the instance, the size of the search space is considerably reduced relatively to the methods that existed previously. The process repeats selecting a variable \( v \) and assigning to it the possible Boolean values and then eliminating all clauses that contain \( v \) or \( \neg v \). The process stops when the instance becomes empty.

The Davis-Putnam-Logemann-Loveland (DPLL)

DPLL [5] is an improved version of DPP w.r.t. the space complexity. Loveland and Logemann introduced the splitting rule, which recursively assigns Boolean values to a variable and solves the resulting sub-problems. When a clause is satisfied, it is suppressed and this operation is called simplification rule.

Other more modern and powerful SAT solvers [10] based on DPLL have been developed and are more widely used these days. They are able to solve instances of hundreds of thousands of clauses over thousands of variables.

As we know, when the size of the instance to solve is very large, the exact algorithms can confront a combinatorial explosion during their execution and any high degree of performance of the actual machines cannot avoid this problem. So "incomplete" approaches have been designed as an alternative to yield near-optimal solutions. Their main drawback is that they never guarantee the optimal solutions. There is nowadays a plethora of techniques, including incomplete heuristics and meta-heuristics. These methods are called incomplete because they explore only a part of the search space when the latter is prohibitive.

Several interesting incomplete SAT solvers have been developed and in some cases, they are even able to outperform the complete solvers. Among them, GSAT [11] is a randomized local search. It starts drawing randomly a valuation for the variables and then making a certain number of flips on variables that reduce the number of unsatisfiable clauses. This process is repeated until getting the optimal solution or reaching a limit on the number of tries.

Walksat [12] is an extended version of GSAT. A noise represented by a probabilistic instruction, is introduced in the procedure to realize the random walk move. With a probability \( p \) a variable drawn at random is considered and with \( (1-p) \) the variable that yields the maximum satisfied clause is selected.

On the other hand, several meta-heuristics like Scatter search, Taboo search and Bio-inspired approaches such as GA, ACO and BSO were adapted for SAT.

3. Association Rules Mining

Association Rules Mining (ARM) is one of the most important and well studied tasks in Data Mining. It aims at extracting frequent patterns, associations or causal structures among sets of items from a given transactional database. Formally, the association rule problem is defined as follows:

let \( T \) be a set of \( n \) transactions \( \{t_1, t_2, \ldots, t_n\} \) representing a transactional database, and \( I \) be a set of \( m \) different items or attributes \( \{i_1, i_2, \ldots, i_m\} \), an association rule is an implication of the form

\[
X \rightarrow Y
\]

where \( X \subseteq I, Y \subseteq I, \) and \( X \cap Y = \emptyset \).

The itemset \( X \) is called antecedent while the itemset \( Y \) is called consequent and the rule is read \( X \) implies \( Y \). It is interpreted as: ‘whenever the items of \( X \) appear in a transaction, the items of \( Y \) appear in the same transaction’.

ARM consists in discovering a set of association rules that cover a large percentage of data transactions. The rules should
not be redundant and hence their size should be kept as small as possible. However, since the databases are increasingly large, the user no longer looks for all the rules but only for a subset of useful rules.

Two basic parameters are commonly used for measuring the usefulness of association rules, namely the support and the confidence of a rule. The support of an itemset \( I' \subseteq I \) is the number of transactions containing \( I' \). The support of a rule \( X \rightarrow Y \) is the support of \( X \cup Y \) and the confidence of a rule is calculated as:

\[
\text{support}(X \cup Y) / \text{support}(X)
\]

The confidence is a measure of strength of the association rule. An association rule \( X \rightarrow Y \) with a confidence of 80% means that 80% of the transactions that contain \( X \) also contain \( Y \) together. So, association rules mining consists in extracting from a given database, all rules with a support greater or equal to \( \text{MinSup} \) and a confidence greater or equal to \( \text{MinConf} \) where \( \text{MinSup} \) and \( \text{MinConf} \) are two thresholds predefined by the users [7].

4. Related Works

For our best knowledge, no studies have tackled the problem of association rules for satisfiability, both paradigms have been explored separately. Pattern recognition such as association rules mining have been widely investigated especially these last years. On the other hand, SAT has been studied for more than half a century. The most recent works that are close to the main concern of the paper are described in the following.

Many algorithms for generating association rules for databases have been proposed in the literature. Some well known algorithms are AIS [1], Apriori [2], Eclat [13] and FP-Growth [8]. AIS is very space consuming and requires too many passes over the whole database. Apriori is the most used algorithm for association rules mining because of its simplicity. It consists of a breadth first search strategy for counting the supports of itemsets and uses a candidate generation function to exploit the downward closure property of support. It presents however a major drawback, which is the great number of access to the database, which yields at its turn a slow computation response. FP-growth uses an efficient but complex structure called \( \text{FP-tree} \) to compress the database and a divide-and-conquer approach, to decompose the mining tasks and the database as well.

In [7] the authors present an interesting survey about different exact and polynomial ARM algorithms. However, because of the fast growth of data, the existing methods have become very quickly inefficient. Indeed, even if these polynomial algorithms can still calculate the association rules in a very short time, they remain limited according to our goal, which is to extract all the rules from large instances of SAT.

Motivated by the diversification aspect of the bio-inspired approach namely the Bee Swarm Optimization (BSO) and the intensification strategy of Taboo Search (TS), a new algorithm called HBSO-TS ( for Hybrid Bees Swarm Optimization and Taboo Search) for association rule mining is proposed. It is applied for generating association rules for SAT clauses and described in the next section.

We propose an approach that starts exploring judiciously the search space before seeking for solutions and hence reduces the complexity of the problem large instances.

5. Hybrid BSO and TS for ARM

In order to achieve scalability for the association rules mining, we designed a new algorithm called HBSO-TS for Hybrid BSO and TS for ARM. We first give an overview of the bio-inspired approach BSO and its corresponding algorithm. Taboo search algorithm is described afterwards. At last the hybrid method is presented.

5.1 BSO

Bee algorithms have been well studied and a good survey on the subject can be found in [9]. They are inspired from the foraging behavior of real bees swarm, which is a cooperative process. The bees start leaving the hive in order to search for a food source. When they found a rich source of nectar, they communicate through a dance, its direction and distance to their congener. This form of communication between the bees is qualified as stygmergic, it allows the bees to share the detected food between themselves.

BSO is based on a population of artificial bees cooperating together to solve an instance of an optimization problem. The general algorithm of BSO is illustrated in Figure 1. In line 2, an initial solution generated randomly or via a heuristic is assigned to the variable representing the reference solution. This solution represent somehow the scout bee, which determines the search regions. The block of instructions between line 3 and line 12 translates the whole bees generations. During one iteration, the determination of the search region to assign to the swarm is performed in line 5. It consists in generating from the reference solution a set of solutions according to some strategy. Each solution in the search region is assigned to one bee of the swarm as starting point of its local search, and the result is stored in a table called \( \text{Dance} \) that the artificial bees use as a means to communicate to their fellows the best solution they found at the end of each iteration. The selection of the reference solution is performed in line 11. It is chosen as the best solution found by the bees in the previous iteration in order to simulate the most rigorous dance performed by a bee.

5.2 Tabu Search

The tabu search we use, is described in Figure 2. It consists in exploring a region intensively. Indeed, in case the algorithm determines a solution \( s \) that outperforms the best solution already found, the best solution will be updated with \( s \). If, on the other hand, the best solution was not improved during
In several iterations, the reference solution will be the one that presents the greatest diversification degree, that is, the one that is the most distant from the solutions of the previous iterations that are stored in the taboo list. This allows the algorithm to escape local optima and ensures a good compromise between exploitation and exploration of the solution space.

Algorithm 1: BSO
1. Begin
2. Generate an initial solution and assign it to refSol;
3. While not stopping criterion do
4. Determine searchPoints from refSol;
5. Assign a solution to each bee;
6. For each Bee do
7. Perform a local search;
8. Store the result in the table
Dance;
9. EndFor;
10. Select the new reference solution RefSol;
11. EndWhile;
12. Return the best solution found;
13. End.

Figure 1. Basic BSO Algorithm

Algorithm 2: Taboo search
1. Begin
2. $S =$ some initial candidate solution;
3. Best = $S$;
4. $L = {};$ a taboo list
5. $I = 1;$
6. while $I < Max-Iter$ and not stop do
7. Enqueue $S$ into $L$;
8. $S =$ Best neighbors($S$);
9. if Quality(Best) < Quality($S$) then Alter Best;
10. EndWhile
11. End

Figure 2. Taboo Search Algorithm

To adapt BSO and TS to ARM we define the following components: the encoding’s solution, the determination of SearchPoints, the fitness function and the neighborhood search.

5.3 Solution Encoding

In Association Rule Mining, two famous representations, namely binary encoding and integer encoding can be found in the literature. In binary encoding, a rule is represented by a vector $S$ of $n$ elements where $n$ is the items number. Furthermore, $S[i] = 1$ if the item $i$ is in the rule and 0 otherwise. In the integer encoding the solution is represented by a vector $S$ of $k+1$ elements where $k$ is the rule size. The first element is the separator index between antecedent and consequent parts of the solution. For all the other elements $i$ in $S$, $S[i] = j$ where the item $j$ appears in the $i$th position of the rule.

In HBSO-TS, a solution is a rule and we propose a combination of both previous representations for it. Such structure is more readable and hence facilitates the design of BSO SearchPoints and the neighborhood search. Each solution $S$ is a vector of $n$ elements where:

1) $S[i] = 0$ if the item $i$ is not in the solution $S$.
2) $S[i] = 1$ if the item $i$ belongs to the antecedent part of the solution $S$.
3) $S[i] = 2$ if the item $i$ belongs to the consequent part.

5.4 Fitness Function

As mentioned above, the ARM problem consists in finding all rules satisfying the constraints of $MinSup$ and $MinConf$ respectively. This is a bi-criteria problem and we can solve it using a multi-objective optimization method. This requires more background and is not the core of our concern. A simple and sufficient solution would be to consider an aggregation technique with two empirical parameters $\alpha$ and $\beta$. The fitness function $F$ of the solution $S$ that we propose is expressed as follows:

$$F(S) = \alpha \times \text{confidence}(S) + \beta \times \text{support}(S)$$

$\alpha$ and $\beta$ are to be set according to the importance we give to the confidence and support respectively.

5.5 Search Regions Determination

Given a reference solution $S_{ref}$ and a colony of $k$ bees, SearchPoints is determined in order to assign a point of the search space to a bee. It is built by changing successively in the solution $S_{ref}$ the values of entries $k+i \times flip$ by random ones drawn from $(0,1,2)$ as defined in subsection 5.3. In order to get several points, we vary $i$ from 0 to $n-1$ and flip is a given parameter. If the distance between solutions is the number of different entries, then the distance between the bees and the solution reference is equal to $\frac{n}{flip}$. The generated points are designed in such a way they are dispersed in the search space in order to browse the latter in an efficient way.

5.6 Neighborhood Search

The neighborhood search concerns BSO for the local search as well as taboo search. A neighbor is computed by changing randomly the value of one element of the current solution. In BSO, the first neighbor that is of better quality than the current solution is considered as the result of the local search. However for the taboo search, all neighbors are generated and evaluated. If the best neighbor does not belong to the taboo list then it will become the new solution for the next iteration. This process is
repeated until the maximum number of taboo search iterations is reached.

5.7 HBSO-TS algorithm

HBSO-TS starts calculating Sref, then it computes k regions to assign to the bees respectively. Then, each bee explores its region using Taboo Search instead of a simple local search and the result is put in Dance table. At this time, the communication between bees takes place in order to find the best solution simulating the most vigorous dance. The latter becomes Sref for the next pass. This process is repeated until a maximum number of iterations is reached.

6. SAT Clauses Preprocessing

HBSO-TS is applied for mining association rules for an instance of SAT. We consider a clause as a transaction and a variable belonging to a clause as its attribute.

6.1 Associative Clusters of Clauses

From the generated association rules, clusters of variables are built. All the clauses appearing in a rule constitute a cluster. The clusters are then formed by the variables belonging to its clusters.

Example:
Let consider the following clauses:
c180 = v10712, v2600, v2510, v9320, v3503, v6178
c551 = v3005, v1718, v9789
c125 = v3572, v4740, v5002, v10585

and the following computed association rule:
C180, c551 → c125
then the variables belonging to these clauses form the following cluster:

6.2 Overall Solution

The resulting clusters are considered as new SAT sub-problems with a smaller number of clauses and variables and hence less difficult to solve. Each cluster is treated separately and the set of clusters may also be solved in parallel. The variables take the value of the solution of the cluster in which they appear. For the resolution of the sub-problems, we used the minisat solver [10]. The overall algorithm is called HBSO-TS-SAT.

This approach would be effective if the clusters were totally disjoints. Nevertheless, this is not the case in general and it happens that several variables are shared by clusters. All the difficulty is then to handle the intersections between clusters. We cope with this issue by adopting a simple strategy, which consists in yielding disjoints clusters by assigning the common variables to only one cluster. Of course, in order to get more convincing outcomes, we should investigate for more efficient strategy for treating the shared variables.

7. Experimental Results

The experiments were conducted on all BMC (Bounded Model Checking) benchmarks [3] presented in Table 1. These instances are hard to solve and used by the scientific community. Extensive experiments were undertaken in order to test the gain brought by the integration of association rules techniques in solving SAT. Our results were compared to those provided by the SAT solver called minisat. The results concerning the satisfaction rate are rather almost as good as those of minisat though the simple way we handle the common variables between clusters. Figure 3 exhibits the results concerning the satisfiability rate of the 13 instances of the tested benchmark. In order to yield better performance, the strategy of solving the shared clauses by clusters should be designed in a more subtle way. Our efforts focus now on this issue.

In terms of computational time, the outcomes are much more interesting. Table 2 shows the gap existing between our approach and the classical version of minisat expressed by the speedup of the last column.

<table>
<thead>
<tr>
<th>benchmark</th>
<th>Number of variables</th>
<th>Number of clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm 1</td>
<td>9685</td>
<td>55870</td>
</tr>
<tr>
<td>ibm 2</td>
<td>2 810</td>
<td>11683</td>
</tr>
<tr>
<td>ibm 3</td>
<td>14930</td>
<td>72106</td>
</tr>
<tr>
<td>ibm 4</td>
<td>28161</td>
<td>139716</td>
</tr>
<tr>
<td>ibm 5</td>
<td>9396</td>
<td>41207</td>
</tr>
<tr>
<td>ibm 6</td>
<td>51639</td>
<td>368352</td>
</tr>
<tr>
<td>ibm 7</td>
<td>8710</td>
<td>39774</td>
</tr>
<tr>
<td>galileo 8</td>
<td>58074</td>
<td>294821</td>
</tr>
<tr>
<td>galileo 9</td>
<td>63624</td>
<td>326999</td>
</tr>
<tr>
<td>ibm 10</td>
<td>59056</td>
<td>323700</td>
</tr>
<tr>
<td>ibm 11</td>
<td>32109</td>
<td>150027</td>
</tr>
<tr>
<td>ibm 12</td>
<td>39598</td>
<td>194778</td>
</tr>
<tr>
<td>ibm 13</td>
<td>13215</td>
<td>65728</td>
</tr>
</tbody>
</table>
results show the impact of clauses clustering gave the best results in terms of processing time. The computation time is reduced drastically because the clusters to solve are of smaller size. The performance of the solution quality is interesting but needs to be improved and this is what we are working on right now.

Also, for the near future, we are interested in exploring several perspectives, among them:
- We intend to deepen our study by handling more experiments on other public benchmarks.
- We are using GPU computing for parallelizing the main algorithm to achieve more enhancements in terms of computation time.
- Another idea is to compute the correlation between the variables in order to reduce their number and integrate this process in the pretreatment phase.

8. Conclusion
In this paper, we proposed an approach for solving problems by clustering the data using an association rules algorithm before launching the resolution method. The satisfiability problem was a case study and algorithms were developed for the problem to illustrate the idea. A new algorithm based on the hybridization of BSO and Taboo search was first designed to address the large-scale association rules mining. Then a novel two-phase approach was designed to solve SAT. The clauses were first grouped using the ARM algorithm and then the clusters were solved separately using the minisat solver. A simple strategy was implemented to handle the shared variables between the clusters.

Experimental tests were performed on a collection of 13 BMC benchmarks to test satisfiability with these novel proposals. The

Table 2. Running time for SAT

<table>
<thead>
<tr>
<th>Instance</th>
<th>Number of rules</th>
<th>CPU Time</th>
<th>Speed up</th>
</tr>
</thead>
<tbody>
<tr>
<td>bmc1</td>
<td>485</td>
<td>0.049</td>
<td>0.065</td>
</tr>
<tr>
<td>bmc2</td>
<td>115</td>
<td>0.0009</td>
<td>0.002</td>
</tr>
<tr>
<td>bmc3</td>
<td>1100</td>
<td>0.21</td>
<td>0.25</td>
</tr>
<tr>
<td>bmc4</td>
<td>12548</td>
<td>0.195</td>
<td>0.212</td>
</tr>
<tr>
<td>bmc5</td>
<td>520</td>
<td>0.385</td>
<td>0.36</td>
</tr>
<tr>
<td>bmc6</td>
<td>14852</td>
<td>0.18</td>
<td>0.25</td>
</tr>
<tr>
<td>bmc7</td>
<td>11005</td>
<td>0.69</td>
<td>0.75</td>
</tr>
<tr>
<td>bmc8</td>
<td>6970</td>
<td>0.12</td>
<td>0.17</td>
</tr>
<tr>
<td>bmc9</td>
<td>8545</td>
<td>0.32</td>
<td>0.26</td>
</tr>
<tr>
<td>bmc10</td>
<td>5200</td>
<td>1.65</td>
<td>1.25</td>
</tr>
<tr>
<td>bmc11</td>
<td>9600</td>
<td>0.3</td>
<td>0.45</td>
</tr>
<tr>
<td>bmc12</td>
<td>18542</td>
<td>3.65</td>
<td>4.52</td>
</tr>
<tr>
<td>bmc13</td>
<td>6578</td>
<td>3.2</td>
<td>4.44</td>
</tr>
</tbody>
</table>

References