Constraint-Based Pattern Discovery

Francesco Bonchi
ISTI-C.N.R., Itay

INTRODUCTION

Devising fast and scalable algorithms, able to crunch huge amount of data, was for many years one of the main goals of data mining research. But then we realized that this was not enough. It does not matter how efficient such algorithms can be, the results we obtain are often of limited use in practice. Typically, the knowledge we seek is in a small pool of local patterns hidden within an ocean of irrelevant patterns generated from a sea of data. Therefore, it is the volume of the results itself that creates a second order mining problem for the human expert. This is, typically, the case of association rules and frequent itemset mining (Agrawal & Srikant, 1994), to which, during the last decade a lot of researchers have dedicated their (mainly algorithmic) investigations. The computational problem is that of efficiently mining from a database of transactions, those itemsets which satisfy a user-defined constraint of minimum frequency.

Recently the research community has turned its attention to more complex kinds of frequent patterns extracted from more structured data: *sequences*, *trees*, and *graphs*. All these different kinds of pattern have different peculiarities and application fields, but they all share the same computational aspects: a usually very large input, an exponential search space, and a too large solution set. This situation—too many data yielding too many patterns—is harmful for two reasons. First, performance degrades: mining generally becomes inefficient or, often, simply unfeasible. Second, the identification of the fragments of interesting knowledge, blurred within a huge quantity of mostly useless patterns, is difficult. The paradigm of *constraint-based pattern mining* was introduced as a solution to both these problems. In such paradigm, it is the user who specifies to the system what is *interesting* for the current application: constraints are a tool to drive the mining process towards potentially interesting patterns, moreover they can be pushed deep inside the mining algorithm in order to fight the exponential search space curse, and to achieve better performance (Srikant et al., 1997; Ng et al. 1998; Han et al., 1999; Grahne et al., 2000).

BACKGROUND

Intuitively the constraint-based pattern mining problem requires to extract from a database the patterns which satisfy a conjunction of constraints. Such conjunction usually contains the minimum frequency constraint, plus other constraints which can be defined on the structure of the patterns (e.g., on the size of the patterns, or on the singletons that the patterns may or may not contain), or on some aggregated properties of the patterns (e.g., the sum of "prices", or the average of "weights").

The following is an example of constraint-based mining query:

$$Q : supp_D(X) \geq 1500 \wedge |X| \geq 5 \wedge avg(X.weight) \leq 15 \wedge sum(X.price) \geq 22$$

it requires to mine, from database D, all patterns which are frequent (have a support at least 1500), have cardinality at least 5, have average weight at most 15 and a sum of prices at least 22.

The constraint-based mining paradigm has been successfully applied in medical domain (Ordonez et al., 2001), and in biological domain (Besson et al., 2005). According to the constraint-based mining paradigm, the data analyst must have a high-level vision of the pattern discovery system, without worrying about the details of the computational engine, in the very same way a database designer has not to worry about query optimization: she must be provided with a set of primitives to declaratively specify to the pattern discovery system how the interesting patterns should look like, i.e., which conditions they should obey. Indeed, the task of composing all constraints and producing the most efficient mining strategy (execution plan) for the given data mining query, should be left to an underlying *query optimizer*. Therefore, constraint-based frequent pattern mining has been seen as a query optimization problem

(Ng et al., 1998), i.e., developing efficient, sound and complete evaluation strategies for constraint-based mining queries.

Among all the constraints, the frequency constraint is computationally the most expensive to check, and many algorithms, starting from Apriori, have been developed in the years for computing patterns which satisfy a given threshold of minimum frequency (see the FIMI repository http://fimi.cs.helsinki.fi for the state-of-the-art of algorithms and softwares). Therefore, a naïve solution to the constraint-based frequent pattern problem could be to first mine all frequent patterns and then test them for the other constraints satisfaction. However more efficient solutions can be found by analyzing the properties of constraints comprehensively, and exploiting such properties in order to push constraints in the frequent pattern computation. Following this methodology, some classes of constraints which exhibit nice properties (e.g., monotonicity, anti-monotonicity, succinctness, etc) have been individuated in the last years, and on the basis of these properties efficient algorithms have been developed.

MAIN RESULTS

A first work defining classes of constraints which exhibit nice properties is Ng et al. (1998). In that work is introduced an Apriori-like algorithm, named CAP, which exploits two properties of constraints, namely *anti-monotonicity* and *succinctness*, in order to reduce the frequent itemsets computation. Four classes of constraints, each one with its own associated computational strategy, are identified:

1. constraints that are anti-monotone but not succinct;
2. constraints that are both anti-monotone and succinct;
3. constraints that are succinct but not anti-monotone;
4. constraints that are neither.

Anti-Monotone and Succinct Constraints

An anti-monotone constraint is such that, if satisfied by a pattern, it is also satisfied by all its subpatterns. The frequency constraint is the most known example of a anti-monotone constraint. This property, *the anti-monotonicity of frequency*, is used by the Apriori (Agrawal & Srikant, 1994) algorithm with the following heuristic: if a pattern X does not satisfy the frequency constraint , then no super-pattern of X can satisfy the frequency constraint, and hence they can be pruned. This pruning can affect a large part of the search space, since itemsets form a lattice. Therefore the Apriori algorithm operates in a level-wise fashion moving bottom-up, level-wise, on the patterns lattice, from small to large itemsets, generating the set of *candidate patterns* at iteration k from the set of frequent patterns at the previous iteration. This way, each time it finds an infrequent pattern it implicitly prunes away all its supersets, since they will not be generated as candidate itemsets. Other anti-monotone constraints can easily be pushed deeply down into the frequent patterns mining computation since they behave exactly as the frequency constraint: if they are not satisfiable at an early level (small patterns), they have no hope of becoming satisfiable later (larger patterns). Conjoining other anti-monotone constraints to the frequency one we just obtain a more selective anti-monotone constraint. As an example, if "price" has positive values, then the constraint $sum(X.price) \leq 50$ is anti-monotone. Trivially, let the pattern X be *{olive_oil, tomato_can, pasta, red_wine}* and suppose that it satisfies such constraints, then any of its subpatterns will satisfy the constraint as well: for instance the set *{olive_oil, red_wine}* for sure will have a sum of prices less than 50. On the other hand, if X does not satisfy the constraint, then it can be pruned since none of its supersets will satisfy the constraint.

Informally, a succinct constraint is such that, whether a pattern X satisfies it or not, can be determined based on the basic elements of X. Succinct constraints are said to be *pre-counting pushable*, i.e., they can be satisfied at candidate-generation time just taking into account the pattern and the single items satisfying the constraint. These constraints are pushed in the level-wise computation by adapting the usual *candidate-generation* procedure of the Apriori algorithm, w.r.t. the given succinct constrain, in such a way that it prunes every pattern which does not satisfy the constraint and that it is not a sub-pattern of any further valid pattern.

Constraints that are both anti-monotone and succinct can be pushed completely in the level-wise computation before it starts (at pre-processing time). For instance, consider the constraint $min(X.price) \geq v$. It is straightforward to see that it is both anti-monotone and succinct. Thus, if we start with the first set of candidates