Exploratory Mining and Pruning Optimizations of Constrained Associations Rules

Raymond T. Ng
University of British Columbia
rng@cs.ubc.ca

Laks V.S. Lakshmanan
Concordia University
laks@cs.concordia.ca

Jiawei Han
Simon Fraser University
han@cs.sfu.ca

Alex Pang
University of British Columbia
cpang@cs.ubc.ca

Abstract

From the standpoint of supporting human-centered discovery of knowledge, the present-day model of mining association rules suffers from the following serious shortcomings: (i) lack of user exploration and control, (ii) lack of focus, and (iii) rigid notion of relationships. In effect, this model functions as a black-box, admitting little user interaction in between. We propose, in this paper, an architecture that opens up the black-box, and supports constraint-based, human-centered exploratory mining of associations. The foundation of this architecture is a rich set of constraint constructs, including domain, class, and SQL-style aggregate constraints, which enable users to clearly specify what associations are to be mined. We propose *constrained association queries* as a means of specifying the constraints to be satisfied by the antecedent and consequent of a mined association.

In this paper, we mainly focus on the technical challenges in guaranteeing a level of performance that is commensurate with the selectivities of the constraints in an association query. To this end, we introduce and analyze two properties of constraints that are critical to pruning: *anti-monotonicity* and *succinctness*. We then develop characterizations of various constraints into four categories, according to these properties. Finally, we describe a mining algorithm called CAP, which achieves a maximized degree of pruning for all categories of constraints. Experimental results indicate that CAP can run much faster, in some cases as much as 80 times, than several basic algorithms. This demonstrates how important the succinctness and anti-monotonicity properties are, in delivering the performance guarantee.

1 Introduction

Since its introduction [1], the problem of mining association rules from large databases has been the subject of numerous studies. These studies cover a broad spectrum of topics including: (i) fast algorithms based on the levelwise Apriori framework [3, 13], partitioning [19, 18], and sampling [24]; (ii) incremental updating and parallel algorithms [6, 2, 8]; (iii) mining of generalized and multi-level rules [21, 9]; (iv) mining of quantitative rules [22, 16]; (v) mining of multi-dimensional rules [7, 14, 12]; (vi) mining rules with item constraints [23]; and (vii) association-rule based query languages [15, 4]. However, from the standpoint of the *user's interaction* with the system, the process of association mining can be summarized as follows. First, the user specifies the part of the database to be mined. Next, the user specifies minimum thresholds for measures such as support and confidence. The system then executes one of several fast mining algorithms. At the end of a highly data intensive process, it returns a very large number of associations, some of which are hopefully what the user was looking for. We contend that there are a number of problems with this present-day model of interaction. Below, we discuss these problems, and highlight our contributions to solving them.

Problem 1 – Lack of User Exploration and Control: Mining (of associations) should be an activity that allows for *exploration* on the user's part [11, 20]. However, the present-day model for mining treats the mining process as an impenetrable black-box – only allowing the user to set the thresholds at the beginning, showing the user all associations satisfying the thresholds at the end, but nothing in between. *What if the user sets the wrong thresholds, or simply wants to change them? What if the user wants to focus the generation of rules to a specific, small subset of candidates, based on properties of the data?*

Such a black-box model would be tolerable if the turnaround time of the computation were small, e.g., a few seconds. However, despite the development of many efficient algorithms [2, 3, 6, 8, 13, 18, 19, 24], association mining remains a process typically taking hours to complete. Before a new invocation of the black-box, the user is not allowed to preempt the process and needs to wait for hours. Furthermore, typically only a small fraction of the computed rules might be what the user was looking for. *Thus the user often incurs a high computational cost that is disproportionate to what the user wants and gets.*

Suggested Principle: (1) Open up the black-box, and establish clear breakpoints so as to allow user feedback. (2) Incorporate user feedback, not only for guidance and control of the mining process, but also for acquiring user's approval for any task involving a substantial cost. [1]

Our Contributions: In Section 2, we present an architecture for exploratory association mining. It divides the black-box into two phases. In Phase I, the user guides the system into finding the intended candidates for the antecedent and consequent of the associations. This can be an iterative process. In Phase II, the user instructs the system to find associations among the selected candidates.

Problem 2 – Lack of Focus: A user may have certain broad phenomena in mind, on which to focus the mining. For example, the user may want to find associations between sets of items whose types do not overlap, or associations from item sets whose total price is under \$100 to items sets whose average price is at least \$1,000 (thereby verifying whether the purchases of cheap items occur together with those of expensive ones). The interface for expressing focus offered by the present-day model is extremely impoverished, because it only allows thresholds for support and confidence to be specified. We note that some recent studies (e.g., [23, 25]) are moving towards the same direction of constraint/query-based association mining and these works will be reviewed below.

Suggested Principle: (3) Provide the user with many opportunities to express the focus. (4) Use the focus to ensure that the system does an amount of computation proportional to what the user gets. This is the first step towards ad hoc mining of associations [11, 20].

Our Contributions: The architecture presented in Section 2 provides a rich interface for the user to express focus. The critical component is the notion of *constrained association queries* (CAQ) to be introduced in Section 3. CAQs offer the user a means for specifying constraints, including domain, class and aggregation constraints, that must be satisfied by the antecedents and consequents of the rules to be mined. Moreover, towards the goal of doing an amount of processing commensurate with the focus specified by the user, we develop in Sections 4-7 various pruning optimizations effected by the constraints. In particular, two key properties, called *anti-monotonicity* and *succinctness*, are introduced and studied. Section 4 analyzes and characterizes all constraints based on *anti-monotonicity*, and Section 5 characterizes constraints based on *succinctness*. Section 6 develops algorithms for mining associations with constraints. One of our algorithms, called CAP, incorporates the earlier analyses on anti-monotonicity and succinctness to support a maximized degree of pruning. Experimental results shown in Section 7 indicate that CAP can run as much as 80 times faster than several algorithms based on the classical Apriori framework.

Srikant et al. consider association mining with item constraints [23]. They consider essentially membership constraints in the context of an item taxonomy, which correspond to a small subclass one of the categories of constraints (see Section 6.3.2) studied in this paper. While we focus on conjunctions and negations of constraints, rather than arbitrary boolean combinations, constraints of the form $S.\texttt{Type} = \textit{sodas}$ do permit implicit forms of disjunction. Besides, we also consider domain and `SQL`-style aggregation constraints, which are far from trivial (cf: Figure 2 later). More importantly, our study focuses on the analysis of the pruning properties of a substantial and naturally useful class of constraints. The classification of constraints based on anti-monotonicity and succinctness is *fundamentally new*. Moreover, Algorithm CAP provides highly effective pruning for much larger classes of constraints than the algorithms given in [23]. Meo et al. propose a language for association mining with conditions [15]. They focus more on sophisticated ways of grouping tuples and do not consider pruning optimizations effected by the conditions. Tsur et al. [25] propose an interesting notion, called "query flocks," for describing parameterized query and filter (a condition applied to the result of the query). However, the filter is confined to lower bound constraints on the number of tuples returned by the query. The general notion of constraints, their classification, and their role in optimization of association mining, fundamentally new in our work, are not studied there.

Problem 3 – Rigid Notion of Relationship: The present-day model restricts the notion of associations to rules with support and confidence that exceed given thresholds. While such associations are useful, other notions of relationships may also be useful. First, there exist several significance metrics other than confidence that are equally meaningful. For example, Brin et al. argue why correlation can be more useful in many circumstances [5]. Second, there may be separate criteria for selecting candidates for the antecedent and consequent of a rule. For example, the user may want to find associations from sets of *items* to sets of *types*. The rule *pepsi* $\Rightarrow$ *snacks* is an instance of such an association, meaning that customers often buy the item *pepsi* together with *any* item of type *snacks*. Coming from different domains, the antecedent and consequent may call for different support thresholds and requirements.

Suggested Principle: (5) Allow the user the flexibility to choose the significance metrics and the criteria to be satisfied by the relationships to be mined.

Our Contributions: The two-phase architecture presented in Section 2 provides this flexibility. Prior to entering Phase II, the user can specify the desired significance metric, and can give different conditions that must be satisfied by the antecedent and consequent of the relationships to be formed.

There are already several proposals in the literature that make the notion of associations less rigid [5, 7, 9, 12, 14, 21]. We are not proposing another here. Instead, we are proposing an architecture that allows many of those alternative notions to co-exist, and that permits the user to choose whatever is appropriate for the application.

2 Architecture

Figure 1 shows a two-phase architecture for exploratory association mining. The user initially specifies a so-called constrained association query, which includes a set of constraints $\mathcal{C}$, including support thresholds for the antecedent and consequent. Each constraint in $\mathcal{C}$ may be applicable to the antecedent, or the consequent, or both. The output of phase I consists of a list of pairs of candidates (S_a, S_c), for the antecedent and consequent satisfying $\mathcal{C}$, such that both S_a and S_c have a support exceeding the thresholds initially

[1] The spirit of this principle is very similar to the principle suggested by Hellerstein et al. for online aggregation [10], in that the user is given the final say as to the amount of information wanted, and is not charged for the cost of computation that is deemed unnecessary by the user.

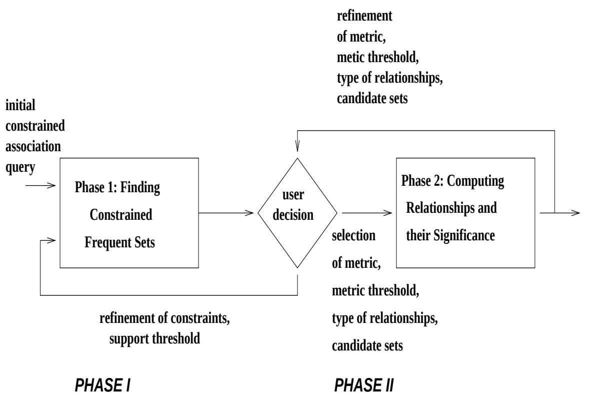


Figure 1: Architecture for Exploratory Association Mining

set by the user. [2] On seeing the candidates, the user can: (i) add, delete, or modify the constraints, and/or (ii) adjust the support thresholds. The user may iterate through Phase I in this manner as many times as desired.

Once satisfied with the current candidate list, the user can instruct the system to proceed to Phase II, wherein the user has the opportunity to specify: (i) the significance metric, (ii) a threshold for the metric specified above, and (iii) whatever further conditions to be imposed on the antecedent and consequent. For instance, if the user wishes to operate in the classical association mining setting, the user would choose confidence as the significance metric, give a confidence threshold, and require that $(S_a \cup S_c)$ be frequent. In this manner, the effort spent in Phase II is geared towards the computation of what the user really wants. Even if Phase II involves costly computation (e.g., computation of correlations), the user has the final say in authorizing such costly operations. Finally, the output of Phase II consists of all associations/relationships that satisfy the conditions given at the beginning of Phase II. Upon examining this output, the user has the opportunity to make further changes to any parameters set before. Depending on which parameters are reset, this may yet trigger Phase I and Phase II, or just Phase II, computations.

A key feature of the proposed architecture is that it is downward compatible. This means that if a user wants only classical associations and the classical mode of interaction, the user can simply set all the appropriate parameters at the beginning, and need not be prompted at the breakpoints for feedback. Of course, we stress that the real power of the architecture stems from its provision for human-centered exploration for association mining, and its implementation of the five principles suggested in the previous section.

The architecture *per se* does not address performance issues. Note that in the computation of classical associations, finding frequent sets is much more expensive than computing rule confidence. Similarly, in our framework, Phase I is computationally much more expensive than Phase II. Thus, in the rest of the paper, we focus on Phase I computations, and designate all Phase II issues as the subject of a forthcoming paper. Regarding Phase I computations, the task is to find all frequent sets satisfying the constraints specified in a constrained association query introduced in the next section. Our technical challenge is to find ways of optimizing the amount of pruning on the computation required in this phase, which is the subject of later sections.

[2] In general the candidate list can be quite large, running in the order of tens of thousands of pairs. An important issue is how to organize such a large list in order to help the user browse through it conveniently. One possibility is to order the list with respect to set inclusion and show only pairs of maximal sets. Another is to provide ranking of the (maximal) sets based on their supports, thus providing feedback to the user as to whether the support thresholds need to be adjusted. This issue is not pursued further in this paper.

$\mathcal{C}$ is a conjunction of constraints on $S1, S2$ drawn from the following classes of constraints.

1. Single Variable Constraints: A single variable (*1-var*) constraint is of one of the following forms.
   (a) *Class Constraint*: It is of the form $S \subset A$, where S is a set variable and `A` is an attribute. It says S is a set of values from the domain of attribute `A`.
   (b) *Domain Constraint*: It is of one of the following forms.
      i. $S\theta v$, where S is a set variable, v is a constant from the domain that S comes from, and θ is one of the boolean operators $=, \neq, <, \leq, >, \geq$. It says *every* element of S stands in relationship θ with the constant value v.
      ii. $v\theta S$, where S, v are as above, and θ is one of the boolean operators $\in, \notin$. This simply says the element v belongs to (or not) the set S.
      iii. $V\theta S$, or $S\theta V$, where S is a set variable, V is a set of constants from the domain S ranges over, and θ is one of $\subseteq, \not\subseteq, \subset, \not\subset, =, \neq$.
   (c) *Aggregate Constraint*: It is of the form $agg(S)\theta v$, where *agg* is one of the aggregate functions *min, max, sum, count, avg*, and θ is one of the boolean operators $=, \neq, <, \leq, >, \geq$. It says the aggregate of the set of numeric values in S stands in relationship θ to v.
2. Two Variable Constraints: A two variable constraint (*2-var*) is of one of the following forms.
   (a) $S_1\theta S_2$, where S_i is a set variable and θ is one of $\subseteq, \not\subseteq, \subset, \not\subset$.
   (b) $(S_1 \circ S_2)\ \theta\ V$, where S_1, S_2 are set variables, V is a set of constants or $\emptyset$, $\circ$ is one of $\cup, \cap$, and θ is one of $=, \neq, \subseteq, \not\subseteq, \subset, \not\subset$.
   (c) $agg_1(S_1)\theta agg_2(S_2)$, where agg_1, agg_2 are aggregate functions, and θ is one of the boolean operators $=, \neq, <, \leq, >, \geq$.

Figure 2: Syntax of Constraint Constructs

3 Constrained Association Queries

As shown in Figure 1, Phase I begins with the user specifying a constrained association query, which states the constraints imposed on the antecedent and consequent of the rules to be mined. To be more precise, the user also needs to specify the part of the database to be mined, called the minable view, and to define the notion of "transactions" for the purpose of calculating support and other metrics. For space limitations, in this paper, we only focus on the specification and optimization of constrained association queries, and do not pursue the issue of how to specify minable view and transactions. We refer the reader to [17] which gives a complete example of how a constrained association query can be "wrapped" in a `SQL` environment. For the purpose of this paper, we may simply regard the minable view as one or a set of relations.

A *constrained association query* (CAQ for short) is defined to be a query of the form: $\{(S_1, S_2) \mid \mathcal{C}\}$, where $\mathcal{C}$ is a set of constraints on S_1, S_2. Note that a CAQ does *not* make the notion of antecedent and consequent of an association explicit. The reason we define a CAQ in this way is that this issue is irrelevant to Phase I. The kinds of constraints studied in this paper are formalized in Figure 2. It includes both single variable constraints – useful in conditioning the antecedent and/or consequent separately, and two variable constraints – useful in constraining them jointly. The constraints cover domain, class, and aggregation constraints. A *set variable* is either an identifier of the form S or is an expression of the form $S.A$, where A is an attribute in the minable view. In addition to the constraints shown in Figure 2, we also include in $\mathcal{C}$ the *frequency constraints* of the form $freq(S_i)$, saying that the support of S_i must exceed some given threshold. The reason for this is that such constraints are pushed inside Phase I computation. Our later analyses on constraints cover frequency constraints as well.

We next illustrate the various constraints of Figure 2 via examples, for which we assume the minable view to be: `trans(TID, Itemset)`, `itemInfo(Item, Type, Price)`. The constraint $S \subset \texttt{Item}$ says that S is a set variable on the `Item` domain. This together with $S.\texttt{Price} \leq 100$ says *all* items in S are of price less than or equal to \$100. $\{\text{snacks}, \text{sodas}\} \subseteq S.\texttt{Type}$ says S should include some items whose type is snacks and some items whose type is sodas, while $S.\texttt{Type} \cap \{\text{snacks}, \text{sodas}\} = \emptyset$ says S should exclude such items. All the above examples are 1-var constraints. Some examples of 2-var constraints are: $S_1.\texttt{Type} \cap S_2.\texttt{Type} = \emptyset$, and $max(S_1.\texttt{Price}) \leq avg(S_2.\texttt{Price})$. The CAQ

$$\{(S_1, S_2) \mid S_1 \subset \texttt{Item} \;\&\; S_2 \subset \texttt{Item} \;\&\; count(S_1) = 1 \;\&\; count(S_2) = 1 \;\&\; freq(S_1) \;\&\; freq(S_2)\}$$

asks for all pairs of single items satisfying frequency constraints. In the examples that follow, for brevity, we suppress the frequency constraints and the domain constraints $S_1 \subset \texttt{Item} \;\&\; S_2 \subset \texttt{Item}$. The CAQ

$$\{(S_1, S_2) \mid agg_1(S_1.\texttt{Price}) \leq 100 \;\&\; agg_2(S_2.\texttt{Price}) \geq 1000\}$$

asks for pairs of item sets, where S_1 has an aggregate price less than \$100 and S_2 has an aggregate price more than \$1,000. Here, agg_1, agg_2 could be any aggregate function given in Figure 2. The CAQ

$$\{(S_1, S_2) \mid count(S_1.\texttt{Type}) = 1 \;\&\; count(S_2.\texttt{Type}) = 1 \;\&\; S_1.\texttt{Type} \neq S_2.\texttt{Type}\}$$

then asks for pairs of item sets of two distinct types. Similarly, the CAQ

$$\{(S_1, S_2) \mid S_1.\texttt{Type} \cap S_2.\texttt{Type} = \emptyset\}$$

asks for pairs of item sets whose associated type sets are disjoint. [3] The CAQ

$$\{(S_1, S_2) \mid S_1.\texttt{Type} \subseteq \{Snacks\} \;\&\; S_2.\texttt{Type} \subseteq \{Beers\} \;\&\; max(S_1.\texttt{Price}) \leq min(S_2.\texttt{Price})\}$$

finds pairs of sets of cheaper snack items and sets of more expensive beer items. The CAQ

$$\{(T_1, T_2) \mid T_1 \subset \texttt{Type} \;\&\; T_2 \subset \texttt{Type}\}$$

finds pairs of sets of types (corresponding to items bought together). Finally, the following CAQ specifies set variables from two different domains.

$$\{(S, T) \mid S \subset \texttt{Item} \;\&\; S.\texttt{Type} = snacks \;\&\; T \subset \texttt{Type} \;\&\; snacks \notin T\}.$$

It asks for sets of snack *items* and non-snack *types*. All the above examples are based on the basket domain. See [17] for examples based on the WWW domain.

Satisfaction of a constraint by a pair of sets (S_1, S_2) is defined in the obvious manner. Satisfaction with respect to a conjunction and negation of constraints is defined in the usual sense as well. Thus, given a CAQ $\equiv \{(S_1, S_2) \mid \mathcal{C}\}$, where $\mathcal{C}$ includes the frequency constraints, we seek to find algorithms that are sound and complete. An algorithm is *sound* provided it only finds frequent sets that satisfy the given constraints; it is *complete* provided all frequent sets satisfying the given constraints are found.

A trivial sound and complete algorithm is to apply the classical Apriori algorithm for finding all frequent sets, and then to test them for constraint satisfaction (called Apriori$^+$ in Section 6). But this algorithm can be highly inefficient. The key technical challenge that we take on is how to guarantee a level of performance (i.e., query processing time) that is commensurate with the selectivities of the constraints in $\mathcal{C}$. The goal is to "push" the constraints as deeply as possible inside the computation of frequent sets. Note that a naive approach to pushing constraints into the Apriori framework of computing frequent sets is to test all candidate sets for constraint satisfaction, before counting for support is conducted. However, this approach is not guaranteed to be complete. Consider the constraint $avg(S.\texttt{price}) \leq 100$. If we apply the usual criterion used by the Apriori Algorithm for pruning away candidate sets for iteration $k+1$, we may miss out certain sets which may potentially be frequent. Specifically, a set S of size $k+1$ may have a subset of size k which was not counted, because the subset did not satisfy the constraint. On the other hand, S itself may well satisfy the constraint. In the following sections, via a comprehensive analysis of the properties of constraints, we develop techniques for "pushing" constraints as deeply in the frequent set computation as possible, while preserving soundness and completeness. Our analysis focuses on pruning optimizations for 1-var constraints only. We have developed pruning optimizations for 2-var constraints as well. But for space limitations, we will only discuss them in a forthcoming paper. Also for space limitations, we suppress several proofs and further discussions on details in this paper. The complete details can be found in the full version [17].

4 Optimization Using Anti-Monotone Constraints

The first property of constraints that we identify and analyze is *anti-monotonicity*. What motivates this property is the observation that the success of the Apriori algorithm for

[3] In the classical setting, there is the implicit condition that S_1 and S_2 are disjoint for the association rule $S_1 \Rightarrow S_2$. We impose the same condition in our framework. However, even when $S_1 \cap S_2 = \emptyset$, it is still meaningful to have such constraints as $S_1.\texttt{Type} = S_2.\texttt{Type}$, $S_1.\texttt{Type} \subseteq S_2.\texttt{Type}$, etc.

1-var Constraint	Anti-Monotone	Succinct
$S\theta v$, $\theta \in \{=, \leq, \geq\}$	yes	yes
$v \in S$	no	yes
$S \supseteq V$	no	yes
$S \subseteq V$	yes	yes
$S = V$	partly	yes
$min(S) \leq v$	no	yes
$min(S) \geq v$	yes	yes
$min(S) = v$	partly	yes
$max(S) \leq v$	yes	yes
$max(S) \geq v$	no	yes
$max(S) = v$	partly	yes
$count(S) \leq v$	yes	weakly
$count(S) \geq v$	no	weakly
$count(S) = v$	partly	weakly
$sum(S) \leq v$	yes	no
$sum(S) \geq v$	no	no
$sum(S) = v$	partly	no
$avg(S)\theta v$, $\theta \in \{=, \leq, \geq\}$	no	no
(frequency constraint)	(yes)	(no)

Figure 3: Characterization of 1-var Constraints: Anti-Monotonicity and Succinctness

classical association mining relies critically on the following property of the frequency constraint: *whenever a set violates the frequency constraint, so does any of its supersets.* Because this property deals with violation of the frequency constraint, it is anti-monotone in nature. [4] This property enables the Apriori algorithm to prune away a significant number of candidate sets that require support counting. The question we ask here is which classes of constraints proposed in the previous section satisfy a similar property. If we can identify such constraints, we can incorporate them in the mining algorithm with the same efficiency with which the conventional frequency constraint is incorporated in the Apriori algorithm. The definition of anti-monotonicity is as follows.

Definition 1 (Anti-monotonicity) A 1-var constraint C is *anti-monotone* iff for all sets S, S':

$$S \supseteq S' \;\&\; S \textit{ satisfies } C \;\Rightarrow\; S' \textit{ satisfies } C.$$

Note that anti-monotonicity is not a new concept. But our main contribution here with respect to this notion is a detailed analysis and a complete characterization of the class of 1-var constraints that are anti-monotone. For space limitations, we summarize in Figure 3 our results for a representative subset of constraints. The figure includes only the positive operators (e.g., $=, \in$) and omits the negative counterparts (e.g., $\neq, \notin$). It also omits the strict operators (e.g., $\subset, <$) and only includes the counterparts with equality (e.g., $\subseteq, \leq$). For reference, it also includes an entry for the frequency constraint.

The second column of the table in Figure 3 identifies which 1-var constraints are anti-monotone. (The third column does the same for succinctness, which will be discussed in Section 5.) Among the domain constraints involving set operators, some are anti-monotone, and some others are not. Similarly, for constraints involving $min()$, $min(S) \geq v$ is anti-monotone, but $min(S) \leq v$ is not. The proof of Theorem 1 below shows why $min(S) \geq v$ is anti-monotone. But for $min(S) \leq v$, even if $min(S) \not\leq v$, it is possible that some superset S' of S may satisfy $min(S') \leq v$. This is because the minimum of a set may decrease when more elements are added into the set. Since $min(S) = v$ is equivalent to $min(S) \geq v$ & $min(S) \leq v$, it is *partly* anti-monotone in the following sense. On the one hand, if S violates $min(S) = v$ because it violates $min(S) \geq v$, then every superset S' of S violates $min(S') = v$, in which case $min(S) = v$ is anti-monotone. On the other hand, if S violates $min(S) = v$ because it violates $min(S) \leq v$, then there may be a superset S' that satisfies $min(S') = v$, in which case $min(S) = v$ is not anti-monotone. Constraints involving $max()$ are "mirror images" of the corresponding $min()$ constraints.

Note that almost all aggregate constraints of the form $agg(S)\theta v$ should be more precisely written as $agg(S.A)\theta v$, where A is a numeric attribute. The only exception are constraints involving $count()$, because both $count(S)\theta v$ and $count(S.A)\theta v$ are meaningful. The former case is represented in the table. The latter case, not duplicated in the table, behaves exactly the same as the corresponding $sum()$ constraints. And regarding $sum()$ constraints, the entries in the figure assume that elements in S are non-negative (e.g., price, length of time). If this assumption is not true, then all forms of constraints involving $sum()$ are not anti-monotone. Finally, all forms $avg()$ constraints are not anti-monotone. We have the following result ascertaining the correctness of the table in Figure 3.

Theorem 1 *For each constraint C listed in the table in Figure 3, C is anti-monotone iff the table says so.*

Proof Sketch. We only show the proof of one case here: $min(S) \geq v$. Proofs of other cases are omitted for brevity. Because $min(S) \not\geq v$, there must exist an element $e \in S$ such that $e < v$. But then the same element e is contained in every superset S' of S. Thus, it is necessary that $min(S') \leq e < v$. ■

Having characterized all anti-monotone 1-var constraints, we next address the issue of how to take full advantage of the anti-monotonicity property of constraints. The basic idea is that any pruning optimization that is applicable to the frequency constraint is also applicable to all anti-monotone constraints. In particular, the standard optimization is to use the following property `(P1)`:

$$S, \textit{ where } |S| = k, \textit{ is frequent} \quad \Longrightarrow \quad \forall S' \subseteq S \textit{ where } |S'| = k-1,\ S' \textit{ is frequent.}$$

In the general setting of processing a constrained association query $\{(S_1, S_2) \mid \mathcal{C}\}$, if $\mathcal{C}_{am}$ consists of all the anti-monotone constraints in $\mathcal{C}$, including the frequency constraint, then an optimization is to use the following property `(P2)`, which generalizes property `(P1)` above.

$$S, \textit{ where } |S| = k, \textit{ satisfies } \mathcal{C}_{am} \quad \Longrightarrow \quad \forall S' \subseteq S \textit{ where } |S'| = k-1,\ S' \textit{ satisfies } \mathcal{C}_{am}.$$

More specifically, if L_{k-1} consists of all the sets of size $k-1$ that satisfy $\mathcal{C}_{am}$, then the set C_k of candidate sets of size k can be generated in exactly the same way as is being done in the Apriori Algorithm. Furthermore, C_k can be further pruned to become C_k^{am} by checking whether each element in C_k satisfies every constraint in $\mathcal{C}_{am}$ other than the

[4] In [13] Mannila et al. refer to this property as monotonicity. But we prefer the term anti-monotone for the reason mentioned above.

frequency constraint. Any element that violates any constraint in $\mathcal{C}_{am}$ is not added to C_k^{am}. Only elements in C_k^{am} require support counting. As will be seen in Section 6.3, this optimization is incorporated in Algorithm CAP.

5 Optimization Using Succinct Constraints

In the previous section, we have identified anti-monotonicity as a useful property of constraints, and characterized the class of 1-var constraints that are anti-monotone. Here we ask the question: can we do better? More specifically, the kind of optimization achievable using anti-monotonicity is restricted to *iterative pruning*: at each iteration or level of the Apriori Algorithm, we can prune the candidate sets which require support counting. However, at each iteration, we still need to generate and test these candidates for satisfaction of the anti-monotone constraints under consideration. Thus, the question we explore here is whether there are classes of constraints for which pruning can be done once-and-for-all before any iteration takes place, thereby avoiding the generate-and-test paradigm. This raises two key questions: (i) Can we succinctly characterize the set of all (item) sets that satisfy a given constraint, and when? (ii) How can we generate all and only those item sets that satisfy the given constraint, avoiding the generate-and-test paradigm completely? Toward (i), we formalize the notion of *succinctness* below. Toward (ii), we propose the notion of a *member generating function*.

In the following, for concreteness, but without loss of generality, we assume that S is a set variable ranging over the domain of attribute `Item`, i.e., $S \subset$ `Item`. We also assume that the minable view consists of the relations: `trans(TID, Itemset)`, `itemInfo(Item, Type, Price)`. Let C be a 1-var constraint. Define $\text{SAT}_C(\texttt{Item})$ to be the set of item sets that satisfy C. With respect to the lattice space consisting of all item sets, $\text{SAT}_C(\texttt{Item})$ represents the *pruned space* consisting of those item sets satisfying C. For example, if $C_1 \equiv S.\texttt{Price} \geq 100$, then the pruned space for C_1 contains precisely those item sets such that each item in the set has a price at least \$100. The notion of a selection predicate will play a useful role in much of the rest of the development. By a *selection predicate*, we mean any predicate allowed to appear as a parameter of a selection operation in relational algebra. In particular, the size of the description of the selection predicate must be *independent* of the size of the database. In the following, for $I \subseteq \texttt{Item}$ and a selection predicate p, we use $\sigma_p(I)$ to denote $\{i \in I \mid \exists t \in \texttt{trans} \bowtie \texttt{itemInfo} : t.\texttt{Item} = i \ \& \ t \text{ satisfies } p\}$. We also use the notation 2^I to mean the strict powerset of I, i.e., the set of all subsets of I except the empty set.

Definition 2 (Succinctness)

1. $I \subseteq \texttt{Item}$ is a *succinct* set if it can be expressed as $\sigma_p(\texttt{Item})$ for some selection predicate p.
2. $SP \subseteq 2^{\texttt{Item}}$ is a *succinct powerset* if there is a fixed number of succinct sets $\texttt{Item}_1, \ldots, \texttt{Item}_k \subseteq \texttt{Item}$ such that SP can be expressed in terms of the strict powersets of $\texttt{Item}_1, \ldots, \texttt{Item}_k$ using union and minus.
3. Finally, a 1-var constraint C is *succinct* provided $\text{SAT}_C(\texttt{Item})$ is a succinct powerset.

For the above constraint $C_1 \equiv S.\texttt{Price} \geq 100$, let $\texttt{Item}_1 = \sigma_{\texttt{Price} \geq 100}(\texttt{Item})$. C_1 is succinct because its pruned space $\text{SAT}_{C_1}(\texttt{Item})$ is simply $2^{\texttt{Item}_1}$. Next consider a more complicated example. Given $C_2 \equiv \{snacks, sodas\} \subseteq S.\texttt{Type}$, the pruned space consists of all those sets that contain at least one item of type *snacks* and at least one item of type *sodas*. Let $\texttt{Item}_2, \texttt{Item}_3, \texttt{Item}_4$ respectively be the sets $\sigma_{\texttt{Type}=`snacks'}(\texttt{Item})$, $\sigma_{\texttt{Type}=`sodas'}(\texttt{Item})$, and $\sigma_{\texttt{Type}\neq`snacks' \wedge \texttt{Type}\neq`sodas'}(\texttt{Item})$. Then, C_2 is succinct because the pruned space $\text{SAT}_{C_2}(\texttt{Item})$ can be expressed as:

$$2^{\texttt{Item}} - 2^{\texttt{Item}_2} - 2^{\texttt{Item}_3} - 2^{\texttt{Item}_4} - 2^{\texttt{Item}_2 \cup \texttt{Item}_4} - 2^{\texttt{Item}_3 \cup \texttt{Item}_4}.$$

We have the following result ascertaining the correctness of the table in Figure 3 characterizing succinct 1-var constraints.

Theorem 2 *For each constraint C listed in the table of Figure 3, C is succinct iff the table says so.*

Proof Sketch. The two examples above illustrate two positive cases. We show the formal proof of one case here: $max(S.A) \geq c$. Formal Proofs of other cases are omitted for brevity. Let $\texttt{Item}_1 = \sigma_{A<c}(\texttt{Item})$. Then the pruned space $\text{SAT}_C(\texttt{Item}) = 2^{\texttt{Item}} - 2^{\texttt{Item}_1}$. The negative cases are proved by producing a pair of witness instances such that (i) no expression for $\text{SAT}_C(\texttt{Item})$ can distinguish between them, but (ii) the sets in $\text{SAT}_C(\texttt{Item})$ corresponding to the two instances are distinct. ■

Consider a constraint of the form $count(S) \leq v$. This constraint is not succinct according to Definition 2, and does not have an MGF of the kind introduced in Definition 3. However, we can have an MGF based on cardinality constraint, i.e., $\{X \mid X \subseteq \texttt{Item} \ \& \ |X| \leq v\}$. Similarly, for $count()$ constraints involving $\theta \in \{=, \geq, \neq, <, >\}$ we can easily set up MGFs with cardinality constraints. Member generation in this manner takes on a flavor different from that introduced earlier. Thus, in the table of Figure 3, we say that $count()$ constraints are *weakly succinct*, to distinguish them from the aforementioned succinct 1-var constraints.

Note that not being succinct has no bearing on the size of the pruned space. For the constraint $C \equiv avg(S.A) = v$, it might well be that $\text{SAT}_C(\texttt{Item})$ is an extremely small subset of $2^{\texttt{Item}}$. But without further concepts and tools, one would be forced to generate arbitrary subsets of `Item` and test them for satisfaction of C. However, in Section 6.3, we will offer a technique that can provide pruning optimization for even such a constraint.

Having characterized the succinct constraints, the immediate question to ask is how to take full advantage of the succinctness property of constraints in pruning. The key concept here is the *member generating function* of a set.

Definition 3 (Member Generating Functions)

1. We say that $SP \subseteq 2^{\texttt{Item}}$ has a *member generating function (MGF)* provided there is a function that can enumerate all and only elements of SP, and that can be expressed in the form $\{X_1 \cup \cdots \cup X_n \mid X_i \subseteq \sigma_{p_i}(\texttt{Item}), 1 \leq i \leq n, \ \& \ \exists k \leq n : X_j \neq \emptyset, 1 \leq j \leq k\}$, for some $n \geq 1$ and some selection predicates $p_1, \ldots, p_n$.

2. A 1-var constraint C is *pre-counting prunable* provided $\text{SAT}_C(\texttt{Item})$ has an MGF.

For the constraint $C_1 \equiv S.\texttt{Price} \geq 100$ discussed above, an MGF is simply $\{X \mid X \subseteq \texttt{Item}_1 \ \&\ X \neq \emptyset\}$, where $\texttt{Item}_1$ is as defined earlier. As for the constraint $C_2 \equiv \{snacks, sodas\} \subseteq S.\texttt{Type}$, an MGF is $\{X_1 \cup X_2 \cup X_3 \mid X_1 \subseteq \texttt{Item}_2 \ \&\ X_1 \neq \emptyset \ \&\ X_2 \subseteq \texttt{Item}_3 \ \&\ X_2 \neq \emptyset \ \&\ X_3 \subseteq \texttt{Item}_4\}$, where $\texttt{Item}_2, \texttt{Item}_3$ and $\texttt{Item}_4$ are as defined earlier.

We have the following lemma showing that succinctness is a sufficient condition for pre-counting prunability. We omit the proof, which is based on an induction on the number of minus operator in the succinctness expression. By considering Theorem 2 and the lemma together, we have identified a large class of constraints that admit pre-counting pruning. In other words, by using the associated MGF, a succinct constraint can simply operate in a generate-only environment – but need not in a generate-and-test environment. Furthermore, the satisfaction of the constraint alone is not affected in any way by the result of the iterative support counting.

Lemma 1 *If C is succinct, then C is pre-counting prunable.*

So far, we have conducted our analysis on a per-constraint basis. But clearly, an algorithm for processing a constrained association query needs to deal with multiple constraints specified in the query. This raises the question of how to produce one MGF for multiple succinct constraints. The following lemma shows how to merge the MGFs of two succinct constraints into one combined MGF.

Lemma 2 *Let C_1, C_2 be two succinct constraints with respective MGFs: $\{S_1 \cup \cdots \cup S_m \mid S_i \subseteq \sigma_{p_i}(\texttt{Item}), 1 \leq i \leq m, \ \&\ \exists m' \leq m : S_i \neq \emptyset, 1 \leq i \leq m'\}$, and $\{T_1 \cup \cdots \cup T_n \mid T_i \subseteq \sigma_{q_i}(\texttt{Item}), 1 \leq i \leq n, \ \&\ \exists n' \leq n : T_i \neq \emptyset, 1 \leq i \leq n'\}$. Then:*
$\{R_{11} \cup \cdots \cup R_{mn} \mid R_{ij} \subseteq \sigma_{p_i \wedge q_j}(\texttt{Item}), 1 \leq i \leq m, 1 \leq j \leq n, \ \&\ R_{k\ell} \neq \emptyset, 1 \leq k \leq m', 1 \leq \ell \leq n'\}$
is an MGF for C_1 & C_2.

While we do not include a proof of the lemma, we use an example to illustrate the construction given above. Consider again the two constraints $C_1 \equiv S.\texttt{Price} \geq 100$ and $C_2 \equiv \{snacks, sodas\} \subseteq S.\texttt{Type}$. Based on the individual MGFs shown above, the combined MGF for both constraints is:

$$\{X_1 \cup X_2 \cup X_3 \mid X_1 \subseteq \sigma_{\texttt{Type}=`snacks' \wedge \texttt{Price} \geq 100}(\texttt{Item}) \ \&\ X_1 \neq \emptyset \ \&\ X_2 \subseteq \sigma_{\texttt{Type}=`sodas' \wedge \texttt{Price} \geq 100}(\texttt{Item}) \ \&\ X_2 \neq \emptyset \ \&\ X_3 \subseteq \sigma_{\texttt{Type} \neq `sodas' \wedge \texttt{Type} \neq `snacks' \wedge \texttt{Price} \geq 100}(\texttt{Item})\}.$$

Combining the MGFs of more than two succinct constraints can be done by straightforward extensions of the construction shown in the above lemma.

6 Algorithms for Computing Constrained Frequent Sets

In this section, we develop algorithms for computing sets that are both frequent and that satisfy the given constraints. In the next section, we will present experimental results comparing them.

```
Algorithm Apriori$^+$
  1 $C_1$ consists of sets of size 1; $k = 1$; $Ans = \emptyset$;
  2 while ($C_k$ not empty) {
      2.1 conduct db scan to form $L_k$ from $C_k$;
      2.2 form $C_{k+1}$ from $L_k$ based on $\mathcal{C}_{freq}$; $k$++; }
  3 for each set $S$ in some $L_k$:
        add $S$ to $Ans$ if $S$ satisfies $(\mathcal{C} - \mathcal{C}_{freq})$.
```

Figure 4: Pseudo Code for Algorithm Apriori$^+$

```
Algorithm Hybrid($m$)
  1 $C_1$ consists of sets of size 1; $k = 1$; $Ans = \emptyset$; $C^* = \emptyset$;
  2 while ($C_k$ not empty and $k \leq m$) {
      2.1 conduct db scan to form $L_k$ from $C_k$;
      2.2 form $C_{k+1}$ from $L_k$ based on $\mathcal{C}_{freq}$; $k$++; }
  3 if $m > 0$ then for each set $S$ in some $L_k$:
        add $S$ to $Ans$ if $S$ satisfies $(\mathcal{C} - \mathcal{C}_{freq})$;
  4 if $C_k$ not empty then {
      4.1 for every $S' \in C_{m+1}$:
            add $S'$ to $C^*$ if $S'$ satisfies $(\mathcal{C} - \mathcal{C}_{freq})$;
      4.2 for every possible $S$ such that for all $S' \subseteq S \wedge (|S'| = m + 1 \Rightarrow S' \in C_{m+1})$:
            add $S$ to $C^*$ if $S$ satisfies $(\mathcal{C} - \mathcal{C}_{freq})$;
      4.3 conduct db scan for sets in $C^*$; add sets that are frequent to $Ans$. }
```

Figure 5: Pseudo Code for Algorithm Hybrid(m)

6.1 Algorithm Apriori$^+$

Let $\mathcal{C}$ denote the set of constraints that a set (of items) must satisfy, one of which is the frequency constraint, denoted by $\mathcal{C}_{freq}$. As illustrated in the computation of classical frequent sets, $\mathcal{C}_{freq}$ provides good pruning (cf: Property P1 in Section 4). Algorithm Apriori$^+$ shown in Figure 4 is a straightforward extension of the classical Apriori Algorithm [3]. It first computes the frequent sets (i.e., Steps 1 and 2), each of which is then verified against the remaining constraints (i.e., Step 3). Ans consists of all sets that satisfy all the constraints $\mathcal{C}$.

6.2 Algorithm Hybrid(m)

Algorithm Apriori$^+$ is not bad if the frequency constraint $\mathcal{C}_{freq}$ is more selective than the remaining ones $(\mathcal{C} - \mathcal{C}_{freq})$. If, however, the converse is true, it is conceivable that checking $(\mathcal{C} - \mathcal{C}_{freq})$ first, followed by verifying $\mathcal{C}_{freq}$, could be better. Algorithm Hybrid(m), shown in Figure 5, with m set to 0, gives such a sequence of executions. With $m = 0$, Steps 2 and 3 are not executed. Then Step 4.2 basically generates and tests all possible sets against $(\mathcal{C} - \mathcal{C}_{freq})$. Those that satisfy the constraints are then counted for support in Step 4.3. Note that Step 4 represents a tradeoff between CPU and I/O costs. On the one hand, Step 4 may generate and test an exponential number of sets, incurring a huge CPU cost. On the other hand, Step 4.2 has the desirable property that only sets that satisfy $(\mathcal{C} - \mathcal{C}_{freq})$ are counted for support. This has the benefit of reducing the amount of database scanning required and hence the I/O cost. This tradeoff could be reasonable if the constraints in $(\mathcal{C} - \mathcal{C}_{freq})$ are highly selective.

To repeat, Apriori$^+$ and Hybrid(0) represent two extremes. The former checks $\mathcal{C}_{freq}$ first, followed by $(\mathcal{C}-\mathcal{C}_{freq})$, whereas the latter checks $(\mathcal{C} - \mathcal{C}_{freq})$ first, followed by $\mathcal{C}_{freq}$. But there are a whole class of algorithms in between the two extremes. Algorithm Hybrid(m) first checks $\mathcal{C}_{freq}$ for m iterations to produce the standard C_{m+1} set – consisting of all candidate sets of size $m + 1$. This takes advantage of the pruning effected by the frequency constraint. Then to reduce the remaining I/O cost, it switches to checking $(\mathcal{C} - \mathcal{C}_{freq})$. In particular, the algorithm only needs to consider a set S whenever all its subsets of size $m + 1$ are in C_{m+1}. This is Step 4.2 in Figure 5. Clearly, the number of candidate sets considered in this step is much less in Algorithm Hybrid(m) than in Algorithm Hybrid(0), and indeed, much less than in Hybrid(n), whenever $n < m$. Thus, compared with Hybrid(0), Hybrid(m) aims to reduce the CPU cost of Step 4.2, at the expense of increasing the I/O cost of Step 2. Conversely, compared with Apriori$^+$, Hybrid(m) aims to reduce the I/O cost of Step 2, at the expense of increasing the CPU cost of Step 4.2. All these algorithms will be experimentally evaluated in the next section.

6.3 Algorithm CAP

While the above algorithms have their own merits, they all fail to exploit the properties of the constraints. In particular, in the preceding sections, we identified anti-monotonicity and succinctness as two useful properties of constraints. In the remainder of this section, we show how Algorithm CAP, to be introduced later, incorporates these ideas by pushing constraints as deep "inside" the computation as possible. We begin by classifying all the constraints into four cases:[5]

I. Constraints that are both anti-monotone and succinct (e.g., $min(S) \geq v$);

II. Constraints that are succinct but not anti-monotone (e.g., $min(S) \leq v$);

III. Constraints that are anti-monotone but not succinct (e.g., $sum(S) \leq v$); and

IV. Constraints that are neither (e.g., $avg(S) \leq v$).

For each category of constraints, we develop a strategy that exploits those constraints to the maximum extent possible.

6.3.1 Succinct and Anti-monotone Constraints

Recall from Definition 3 the general form for an MGF for $\text{SAT}_C(\texttt{Item})$, when C is succinct. It can be shown that when C is anti-monotone as well, then the MGF must be of the form $\{S \mid S \subseteq \sigma_p(\texttt{Item}) \ \& \ S \neq \emptyset\}$ [17]. Then the set C_1 of candidate sets of size 1 in the Apriori Algorithm can simply be replaced by $C_1^c = \{e \mid e \in C_1 \ \& \ e \in \sigma_p(\texttt{Item})\}$. Furthermore, since C is anti-monotone, sets containing any element *not in* C_1^c need *not* be considered. Thus, we have the following strategy and lemma for succinct anti-monotone constraints.

Strategy I:

- Replace C_1 in the Apriori Algorithm by C_1^c defined above.

[5]For brevity, we do not discuss pruning induced by weakly succinct constraints in detail here. Essentially, these constraints contribute new starting and/or termination conditions for the basic levelwise algorithm.

We have the following formal result.

Lemma 3 *Let C be a succinct anti-monotone constraint. Then strategy I above is sound and complete w.r.t. determining the set of all frequent sets that satisfy C.*

6.3.2 Succinct but Non-anti-monotone Constraints

Since the constraint is not anti-monotone, the challenge in this case is that we cannot guarantee that all sets satisfying the constraint must be subsets of C_1^c. Fortunately, we can make use of the structure given by the MGF of the constraint. Here for brevity, we only describe the strategy for the case where the MGFs are of the simple form $\{S_1 \cup S_2 \mid S_1 \subseteq \sigma_{p_1}(\texttt{Item}) \ \& \ S_1 \neq \emptyset \ \& \ S_2 \subseteq \sigma_{p_2}(\texttt{Item})\}$.

Strategy II:

- Define $C_1^c = \sigma_{p_1}(\texttt{Item})$ and $C_1^{\neg c} = \sigma_{p_2}(\texttt{Item})$. Define corresponding sets of frequent sets of size 1: $L_1^c = \{e \mid e \in C_1^c \ \& \ \mathit{freq}(e)\}$, and $L_1^{\neg c} = \{e \mid e \in C_1^{\neg c} \ \& \ \mathit{freq}(e)\}$.
- Define $C_2 = \{\{e, f\} \mid e \in L_1^c \ \& \ f \in (L_1^c \cup L_1^{\neg c})\}$, and L_2, as usual, the set of frequent sets in C_2, i.e., $L_2 = \{S \mid S \in C_2 \ \& \ \mathit{freq}(S)\}$.
- In general, C_{k+1} can be obtained from L_k in exactly the same way as in the Apriori Algorithm – with only one modification. In the classical case, a set S is in C_{k+1}, if all its subsets of size k are in L_k, i.e.,

  $$\forall S' : \ S' \subset S \text{ and } |S'| = k \Rightarrow \ S' \in L_k.$$

  In the case of succinct, non-anti-monotone constraints, we only need to check subsets S' that intersect with L_1^c. In other words, subsets S'' that are disjoint with L_1^c are excluded in the above verification. These are sets that only contain elements from $L_1^{\neg c}$, and that are not counted for support. Because we do not know whether they are frequent or not, we give them the benefit of the doubt. Hence, we have the following modification in candidate generation. A set S is in C_{k+1}, if for all subsets:

  $$\forall S' : \ S' \subset S \text{ and } |S'| = k \text{ and } S' \cap L_1^c \neq \emptyset \\ \Rightarrow \ S' \in L_k.$$

  As usual, $L_{k+1} = \{S \mid S \in C_{k+1} \ \& \ \mathit{freq}(S)\}$.

For example, suppose that $\texttt{Item}$ is the set $\{1, \ldots, 50\}$, C_1^c is $\{1, \ldots, 20\}$, and $C_1^{\neg c}$ is $\{21, \ldots, 50\}$. Further suppose that L_1^c turns out to be $\{1, \ldots, 10\}$, and that $L_1^{\neg c}$ turns out to be $\{41, \ldots, 50\}$. Then C_2 corresponds to the Cartesian product $\{1, \ldots, 10\} \times \{1, \ldots, 10, 41, \ldots, 50\}$. This gives all the sets of size 2 that can be frequent and that contain at least one element from C_1^c. Suppose that $\{1, 41\}$ and $\{1, 42\}$ are frequent (i.e., $\in L_2$) and that $\{1, 43\}$ is not. Then in considering whether $\{1, 41, 42\}$ should be in C_3, we only check whether $\{1, 41\}$ and $\{1, 42\}$ are frequent, with the benefit of the doubt given to $\{41, 42\}$, whose support frequency is unknown. Thus, $\{1, 41, 42\}$ is in C_3. But neither $\{1, 41, 43\}$ nor $\{1, 42, 43\}$ is in C_3, because $\{1, 43\}$ is not frequent. In this example, it is not difficult to show that any frequent set satisfying the constraint (i.e., containing at least one element from C_1^c) will be counted and verified to be frequent under Strategy II. In general, we have the following result, which can be proved by induction on k, the size of the sets.

Lemma 4 *Let C be a constraint that is succinct but not anti-monotone. Strategy II proposed above is sound and complete w.r.t. computing the frequent sets that satisfy C.*

So far, we have only considered the MGF of the simple form given above. Strategy II and Lemma 4 above can be generalized to deal with the general form of MGF given in Definition 3. For space limitations, we omit the details but give the following example. For the constraint $C_2 \equiv \{snacks, sodas\} \subseteq S.\mathtt{Type}$ discussed in Section 5, the MGF is $\{X_1 \cup X_2 \cup X_3 \mid X_1 \subseteq \mathtt{Item}_2 \ \&\ X_1 \neq \emptyset \ \&\ X_2 \subseteq \mathtt{Item}_3 \ \&\ X_2 \neq \emptyset \ \&\ X_3 \subseteq \mathtt{Item}_4\}$, where $\mathtt{Item}_2, \mathtt{Item}_3$ and $\mathtt{Item}_4$ are as defined earlier. Corresponding to X_i, the strategy is to first form the sets $L_1^{X_i}$, as $L_1^{X_i} = \{e \mid e \in \mathtt{Item}_{i+1} \ \&\ \mathit{freq}(\{e\})\}$, $1 \leq i \leq 3$. Then we form the candidate set $C_2 = L_1^{X_1} \times L_1^{X_2}$, from which the set L_2 is obtained. Next we form $C_3 = L_2 \times (L_1^{X_1} \cup L_1^{X_2} \cup L_1^{X_3})$. From this point onwards, L_k and C_{k+1} are computed as usual, with the only modification in candidate generation that S is in C_{k+1}, iff: for all subsets $S' \subset S$ and $|S'| = k$ and $S' \cap L_1^{X_1} \neq \emptyset$ and $S' \cap L_1^{X_2} \neq \emptyset \Rightarrow S' \in L_k$. Complete details can be found in the full paper [17].

6.3.3 Anti-monotone but Non-succinct Constraints

Because constraint C is not succinct, it does not admit an MGF that can be used to avoid the generate-and-test paradigm completely. However, because of anti-monotonicity, whenever candidate sets for a particular level are generated, satisfaction of C is tested first – *before* counting is done. Those sets not satisfying C can be dropped right away.

Strategy III:

- Define C_k as in the classical Apriori Algorithm. Drop a set $S \in C_k$ from counting if S fails C, i.e., constraint satisfaction is tested *before* counting is done.

Lemma 5 *Let C be a constraint that is anti-monotone, but not succinct. Then Strategy III above is sound and complete w.r.t. computing the frequent sets that satisfy C.*

6.3.4 Non-succinct and Non-anti-monotone Constraints

Our methodology for dealing with a constraint C that is neither anti-monotone nor succinct, is to try and *induce weaker constraints* that might be anti-monotone and/or succinct. If we can find such constraints, then the latter can be exploited using one of the strategies outlined above. However, in this case, once the frequent set determination algorithm terminates, one final round of testing the frequent sets for satisfaction of C is necessary. The reason is that the frequent sets, while guaranteed to satisfy the induced weaker constraints, may not necessarily satisfy C.

Strategy IV:

- Induce any weaker constraint C' from C. Depending on whether C' is anti-monotone and/or succinct, use one of the strategies I-III above for the generation of frequent sets.

Algorithm CAP

```
1 if $\mathcal{C}_{sam} \cup \mathcal{C}_{suc} \cup \mathcal{C}_{none}$ is non-empty, prepare $C_1$ as indicated in Strategies I, II and IV; k = 1;
2 if $\mathcal{C}_{suc}$ is non-empty {
    2.1 conduct db scan to form $L_1$ as indicated in Strategy II;
    2.2 form $C_2$ as indicated in Strategy II; k = 2;}
3 while ($C_k$ not empty) {
    3.1 conduct db scan to form $L_k$ from $C_k$;
    3.2 form $C_{k+1}$ from $L_k$ based on Strategy II if $\mathcal{C}_{suc}$ is non-empty, and Strategy III for constraints in $\mathcal{C}_{am}$;
    }
4 if $\mathcal{C}_{none}$ is empty, $Ans = \bigcup L_k$. Otherwise, for each set $S$ in some $L_k$, add $S$ to $Ans$ iff $S$ satisfies $\mathcal{C}_{none}$.
```

Figure 6: Pseudo Code for Algorithm CAP

- Once all frequent sets are generated, test them for satisfaction of C.

For example, let C be the constraint $avg(S.A) \leq v$. Although C is neither anti-monotone nor succinct, it induces the weaker constraint $C' \equiv min(S.A) \leq v$. More precisely, every set S that satisfies C necessarily satisfies C'. Since C' is a succinct but non-anti-monotone constraint, Strategy II can be applied. However, once all frequent sets are generated, each of them should be tested for satisfaction of C. It should be straightforward to see that Strategy IV is sound and complete.

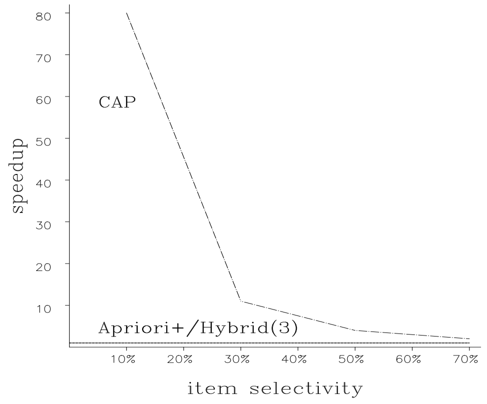
6.3.5 Handling Multiple Constraints

So far, our analysis has concentrated on individual constraints. An important question is, given a set of constraints in a constrained association query, each corresponding to one of the four categories analyzed above, how they can be incorporated in an algorithm. Let $\mathcal{C}$ be the set of constraints in a CAQ, including the frequency constraints, and let $\mathcal{C}_{sam}$ (resp., $\mathcal{C}_{suc}$, $\mathcal{C}_{am}$, and $\mathcal{C}_{none}$) denote those constraints among $\mathcal{C}$ that are succinct and anti-monotone (resp., that are succinct but not anti-monotone, that are anti-monotone but not succinct, and that are neither of the above). We assume without loss of generality that weaker constraints induced from those in $\mathcal{C}_{none}$ are included in $\mathcal{C}_{suc}$, $\mathcal{C}_{am}$, or $\mathcal{C}_{none}$, depending on the nature of the constraints induced.

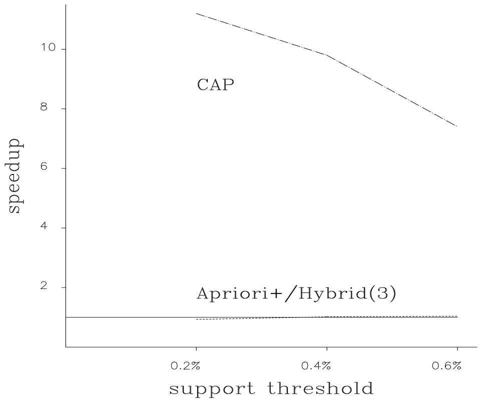
Strategy for Handling Multiple Constraints:

- Combine the MGFs of all the constraints in $\mathcal{C}_{sam}$ as shown in Lemma 2. Apply Strategy I with the combined MGF.
- Combine the MGFs of all the constraints in $\mathcal{C}_{suc}$. Apply Strategy II with the combined MGF.
- For all constraints in $\mathcal{C}_{am}$, follow Strategy III.
- For each constraint in $\mathcal{C}_{none}$, induce weaker constraints and apply Strategy IV.

As depicted in Figure 6, Algorithm CAP ("Constrained APriori") incorporates the strategy for handling multiple



(a) Speedup vs Selectivity



(b) Speedup vs Support

Figure 7: Performance Comparison for Succinct and Anti-monotone Constraints

constraints, and shows where the various strategies are applied within the Apriori framework, thus pushing the constraints in a CAQ deep inside the mining process.

7 Experimental Evaluation

To evaluate the relative efficiency of the various algorithms presented in Section 6, we implemented all of them in C. We used the program developed at IBM Almaden Research Center [3] to generate the transactional databases. While we experimented with various databases, the results cited below are based on a database of 100,000 records and a domain of 1000 items. The page size was 4Kbytes. Experiments were run in a SPRAC-10 environment.

(1) In the first set of experiments, we compare the algorithms using the succinct and anti-monotone constraint $max(S.\texttt{Price}) \leq v$. Different values of v correspond to different selectivities of the constraint. In Figure 7(a), an x-% selectivity means that there are x-% of the items whose price is $\leq v$, whatever that value may be. The y-axis shows the speedup of various algorithms relative to Algorithm Apriori$^+$, i.e., the execution time of Apriori$^+$ divided by that of the algorithm being considered.

Figure 7(a) plots the speedup as a function of constraint selectivity, with support threshold set at 0.5%. It clearly shows the dominance of Algorithm CAP compared with Apriori$^+$. For instance, for a 10% selectivity, CAP runs 80 times faster than Apriori$^+$! Even for a 30% selectivity, the speedup is about 10 times.

Hybrid(0), Hybrid(1) and Hybrid(2) all take much too long when compared with Apriori$^+$. This clearly demonstrates that the CPU cost of finding all sets that satisfy the constraints in a generate-and-test mode is prohibitive. Using the MGF to generate all such sets is much better. Hybrid(3), Hybrid(4), etc. take more or less the same time as Apriori$^+$, almost always within 5% of each other.[6] Recall that the Hybrid(m) algorithms try to reduce database scanning time at the expense of increasing CPU time. It is quite possible that when the transaction database is larger in size, the Hybrid(m) algorithms will become more superior to Apriori$^+$. However, it is highly unlikely that they will ever catch up with Algorithm CAP. The Hybrid(m) algorithms will be omitted in subsequent discussions.

Figure 7(a) shows the comparisons of the algorithms when the support threshold is set to 0.5%. Figure 7(b) shows the comparisons when the support threshold varies, but with the item selectivity fixed at 30%. When the support threshold is low, Apriori$^+$ need to find and process a lot of frequent sets, most of which turn out not to satisfy the constraint. When the support threshold is high, the total number of frequent sets decreases, and Apriori$^+$ behaves relatively better. Still, it processes too many frequent sets violating the constraint, and CAP is at least 8 times faster.

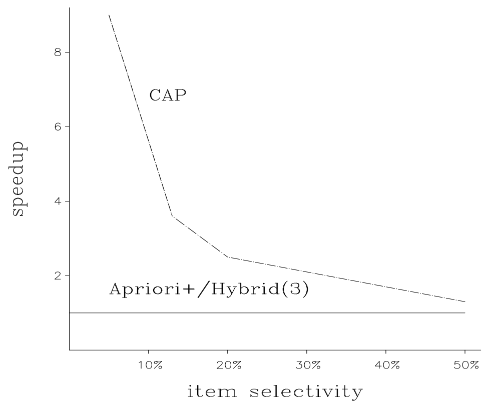
The effectiveness of the pruning achieved by CAP is best captured by the following tables.

support	L_1	L_2	L_3	L_4
0.2%	174/582	79/969	29/1140	8/1250
0.6%	98/313	1/12	0/1	0

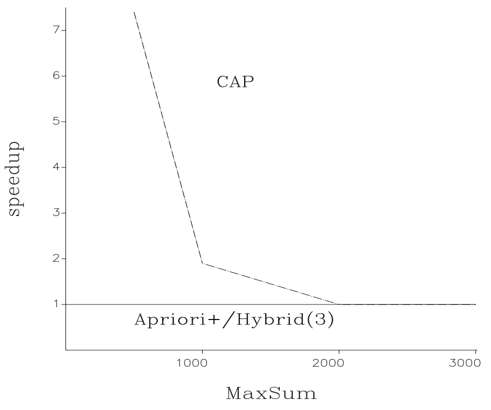
support	L_5	L_6	L_7	L_8
0.2%	1/934	0/451	0/132	0/20
0.6%	0	0	0	0

Based on a setting that is the same as that shown in Figure 7(b), the two rows correspond to support thresholds of 0.2% and 0.6% respectively. The columns correspond to the sizes of the frequent sets. Each entry is of the form a/b, where a is the number of frequent sets satisfying the constraint, and b is simply the total number of frequent sets of that size. For example, for a support of 0.2% and sets of size 4, Apriori$^+$ finds 1,250 frequent sets, only 8 of which need to be found by CAP. Furthermore, Apriori$^+$ finds frequent

[6]Hybrid(m) for $m \geq 4$ incurs the same I/O in the first several iterations as Apriori$^+$. Savings on I/O corresponding to later iterations will be dominated by the heavy I/O incurred in the first few iterations. This point is reinforced by our experiments.



(a) Succinct, Non-anti-monotone (Non-succinct, Non-anti-monotone)



(b) Anti-monotone, Non-succinct

Figure 8: Performance Comparison for the Other Three Types of Constraints

sets of size 6 and up, whereas CAP correctly stops after size 5. We choose the support thresholds such that Apriori$^+$ requires between 5 and 8 iterations. We can conclude: (i) if the support thresholds are chosen to be lower corresponding to more iterations by Apriori$^+$, the overall speedup of CAP would be even more pronounced. (ii) Even if the support thresholds are chosen to be higher so the Apriori$^+$ finishes in fewer than 5 iterations, the speedup achieved by CAP would still be substantial. The reason is that CAP achieves considerable savings in the very first few iterations, compared to Apriori$^+$.

(2) In the next series of experiments, we use the succinct but non-anti-monotone constraint $\{soda\} \subseteq S.\texttt{Type}$. In Figure 8(a), an x-% selectivity means that there are x-% of the items whose type is *soda*. The y-axis shows the speedup. For selectivities of 5%, 10% and 20%, CAP runs 9, 4 and 2.5 times faster than Apriori$^+$ respectively.

When compared with the previous case, i.e., succinct and anti-monotone constraints, the gain here shown by CAP comes entirely from the succinctness of the constraint. This shows the importance of taking advantage of the succinctness property for achieving pruning. Furthermore, for a given selectivity, a constraint that is both succinct and anti-monotone effects much more pruning than a constraint that is only succinct. This shows that succinctness and anti-monotonicity produce a powerful compound pruning effect.

(3) Next, we consider the constraint $Avg(S.\texttt{Price}) \leq v$. This constraint by itself is tough to optimize because it is neither succinct nor anti-monotone. But as shown before, the constraint induces the weaker constraint $min(S.\texttt{Price}) \leq v$, which is succinct but non-anti-monotone. The extra cost is that a final verification step is needed to check whether the frequent sets really satisfy $Avg(S.\texttt{Price}) \leq v$. As it turns out, the comparison between CAP and Apriori$^+$ for $Avg(S.\texttt{Price}) \leq v$ is almost identical to the comparison for a typical succinct, non-anti-monotone constraint. Thus, Figure 8(a) applies. This shows that the final verification step takes a relatively small amount of effort. This also shows the effectiveness of inducing weaker constraints, and pushing them inside to achieve pruning.

(4) Finally, we consider the last category of constraints – anti-monotone but not succinct. The constraint used is $sum(S.\texttt{Price}) \leq MaxSum$. Unlike the previous cases, the notion of item selectivity does not directly make sense in this case. Instead, we do the following. The 1,000 items in our domain are numbered from 1 to 1,000. The `Price` value of each item is exactly its number, i.e., from 1 to 1,000. Then $MaxSum$ is set to be 500, 1000, 2000 and so on. This gives the x-axis of Figure 8(b). Contrary to the other cases, the dominance of CAP drops drastically as the value of $MaxSum$ increases. In fact, beyond $MaxSum$ = 2000, CAP shows no gain over Apriori$^+$. The following tables illustrate why.

$MaxSum$	L_1	L_2	L_3	L_4
500	184/362	16/66	4/91	0/105
1000	362/362	39/66	28/91	15/105
2000	362/362	66/66	86/91	77/105

$MaxSum$	L_5	L_6	L_7	L_8
500	0/77	0/35	0/9	0/1
1000	1/77	0/35	0/9	0/1
2000	40/77	13/35	1/9	0/1

When $MaxSum$ is 500, the constraint helps reduce the number of frequent sets at level 1 (i.e., size 1) by a half (i.e., 184 out of 362). This reduction is compounded at subsequent levels. As shown in Figure 8(b), this gives rise to a speedup of over 7 times. When $MaxSum$ is increased to 1,000, there is no pruning at level 1 (i.e., other than the pruning of the frequency constraint). It is only at level 2 that the constraint provides some pruning (i.e., 39 out of 66). This modest reduction is compounded at later levels, giving an overall speedup of about 2 times. But when $MaxSum$ is increased to 2,000, there is no pruning at the first two levels. Even though there is some pruning later on, the pruning comes too little too late. The key observation is that for any pruning optimization to be significant, it has to be effective *as early as possible* at the first two levels. Any reduction obtainable earlier is compounded to give a significant speedup subsequently. This observation explains convincingly why in previous cases, succinctness can help produce such an impressive speedup for CAP.

8 Conclusions and Future Work

Motivated by the problems – (i) lack of user exploration and control, (ii) lack of focus, and (iii) rigid notion of relationships – with the present-day model of association mining from very large databases, we first propose a model whereby the user specifies a constrained association query to the system. This allows the user to tell the system to focus the mining task on associations that satisfy the constraints specified, which include class, domain, and aggregation constraints. The mining process is structured using a two-phase architecture which provides numerous opportunities for user feedback, control, and approval. Second, and more importantly, to deliver on the goal that the performance must be commensurate with the focus expressed by the user via constraints, we develop techniques for pushing the constraints deep inside the mining process, to achieve a maximized degree of pruning in the amount of processing required. Pivotal to this are the *anti-monotonicity* and *succinctness* properties, based on which we characterize constraints into various categories. From our characterizations, we develop and discuss several algorithms for computing constrained queries, and evaluate their relative efficiency with a series of experiments. Among those, Algorithm CAP, which exploits constraints as early as possible in the mining process, achieves a remarkable speedup.

Several questions remain open. (1) For 1-var constraints that are neither anti-monotone nor succinct, we advocate the methodology of inducing weaker constraints, which may be anti-monotone and/or succinct. What is the method for inferring these induced constraints? And how to find the constraints that optimize pruning? (2) For space limitations, we only discuss 1-var constraints in this paper. We have already carried out a detailed analysis of 2-var constraints. Our methodology for pushing 2-var constraints as well as related algorithms will be discussed in a forthcoming paper. (3) The entire paper has focused on computations related to Phase I of our architecture. Phase II is concerned with forming associations, based on the user specified notion of significance metric and type of relationship, and is orthogonal to the contributions of this paper. This will be explored in a future paper. (4) Finally, our eventual goal is to develop a framework for *ad hoc mining*, whereby a user can interact with a mining system in much the same way that the user does with an RDBMS today. Many believe that this is crucial for the technology of database mining to reach its full potential (see also [11]). We believe this paper takes an important step in this direction.

References

[1] R. Agrawal, T. Imielinski and A. Swami. Mining association rules between sets of items in large databases. SIGMOD 93, pp 207–216.

[2] R. Agrawal and J. C. Shafer. Parallel mining of association rules: Design, implementation, and experience. IEEE TKDE, 8, pp 962–969, Dec 1996.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. VLDB 94, pp 487–499.

[4] E. Baralis and G. Psaila. Designing templates for mining association rules. Journal of Intelligent Information Systems, 9, pp 7–32, 1997.

[5] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. SIGMOD 97, pp 265–276.

[6] D.W. Cheung, J. Han, V. Ng, and C.Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. ICDE 96, pp 106–114.

[7] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining using two-dimensional optimized association rules: Scheme, algorithms, and visualization. SIGMOD 96, pp 13–23.

[8] E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. SIGMOD 97, pp 277–288.

[9] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. VLDB 95, pp 420–431.

[10] J. Hellerstein, P. Haas and H. Wang. Online Aggregation. SIGMOD 97, pp 171–182.

[11] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. Communications of ACM, 39, pp 58–64, 1996.

[12] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. KDD 97, pp 207–210.

[13] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. CIKM 94, pp 401–408.

[14] B. Lent, A. Swami, and J. Widom. Clustering association rules. ICDE 97.

[15] R. Meo, G. Psaila, and S. Ceri. A new SQL-like operator for mining association rules. VLDB 96, pp 122–133.

[16] R.J. Miller and Y. Yang. Association rules over interval data. SIGMOD 97, pp 452–461.

[17] Raymond T. Ng, Laks V.S. Lakshmanan, Jiawei Han, and Alex Pang. Exploratory Mining and Optimization of Constrained Association Queries. Technical Report, University of British Columbia and Concordia University, October 1997.

[18] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. SIGMOD 95, pp 175–186.

[19] A. Savasere, E. Omiecinski, and S. Navathe. An efficient algorithm for mining association rules in large databases. VLDB 95, pp 432–443.

[20] A. Silberschatz and S. Zdonik. Database systems – breaking out of the box. SIGMOD Record, 26, pp 36–50, 1997.

[21] R. Srikant and R. Agrawal. Mining generalized association rules. VLDB 95, pp 407–419.

[22] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. SIGMOD 96, pp 1–12.

[23] R. Srikant, Q. Vu and R. Agrawal. Mining association rules with item constraints. KDD 97, pp 67–73.

[24] H. Toivonen. Sampling large databases for association rules. VLDB 96, pp 134–145.

[25] D. Tsur, J. D. Ullman, S. Abitboul, C. Clifton, R. Motwani, and S. Nestorov. Query flocks: A generalization of association-rule mining. *http://www-db.stanford.edu/∼ullman/pub/qflocks.ps*, 1–20, Oct. 1997.