

# Optimization of Constrained Frequent Set Queries with 2-variable Constraints

**Laks V.S. Lakshmanan**

IIT Bombay

laks@math.iitb.ernet.in

(Currently on leave from Concordia U.)

**Raymond Ng**

U. of British Columbia

rng@cs.ubc.ca

**Jiawei Han**

Simon Fraser U.

han@cs.sfu.ca

**Alex Pang**

U. of British Columbia

cpang@cs.ubc.ca

## Abstract

Currently, there is tremendous interest in providing ad-hoc mining capabilities in database management systems. As a first step towards this goal, in [15] we proposed an architecture for supporting constraint-based, human-centered, exploratory mining of various kinds of rules including associations, introduced the notion of constrained frequent set queries (CFQs), and developed effective pruning optimizations for CFQs with 1-variable (1-var) constraints.

While 1-var constraints are useful for constraining the antecedent and consequent separately, many natural examples of CFQs illustrate the need for constraining the antecedent and consequent jointly, for which 2-variable (2-var) constraints are indispensable. Developing pruning optimizations for CFQs with 2-var constraints is the subject of this paper. But this is a difficult problem because: (i) in 2-var constraints, both variables keep changing and, unlike 1-var constraints, there is no fixed target for pruning; (ii) as we show, “conventional” monotonicity-based optimization techniques do not apply effectively to 2-var constraints.

The contributions are as follows. (1) We introduce a notion of *quasi-succinctness*, which allows a quasi-succinct 2-var constraint to be reduced to two succinct 1-var constraints for pruning. (2) We characterize the class of 2-var constraints that are quasi-succinct. (3) We develop heuristic techniques for non-quasi-succinct constraints. Experimental results show the effectiveness of all our techniques. (4) We propose a query optimizer for CFQs and show that for a large class of constraints, the computation strategy generated by the optimizer is *ccc-optimal*, i.e., minimizing the effort incurred w.r.t. constraint checking and support counting.

## 1 Introduction

Since the introduction of association rules [1], the development of effective mechanisms for mining large databases has been the subject of numerous studies, which can be broadly

divided into two groups. The first group includes studies focusing on performance and efficiency issues, e.g., the Apriori framework [2, 11], partitioning [16], sampling [24], incremental updating [6], etc. The second group includes studies that go beyond the initial notion of association rules to other kinds of mined rules, e.g., multi-level rules [8, 21], quantitative and multi-dimensional rules [22, 7, 14, 10], rules with item constraints [23], mining long patterns [3], correlations and causal structures [4, 20], ratio rules [12], etc.

Recently it has been recognized that the integration of data mining technologies with database management systems is of crucial importance [5]. Furthermore, it has been argued that the fundamental distinction of a data mining system from a statistical analysis program or a machine learning system *should* be that the former: (i) offers an ad-hoc mining query language and (ii) supports efficient processing and optimization of mining queries [9, 19]. Sarawagi et al. [18] study the suitability of different architectures for the integration of association mining with DBMS and study the relative performance tradeoffs. Tsur et al. [25] explore the question of how techniques like the well-known Apriori algorithm can be generalized beyond their current applications to a generic paradigm called query flocks. While these are important results toward enabling the integration of association mining and DBMS, we contend that *ad-hoc mining still cannot be supported until the following fundamental problems in the present-day model of mining, first identified in [15], are addressed satisfactorily*: (i) lack of user exploration and guidance (e.g., expensive computation undertaken without user’s approval), and (ii) lack of focus (e.g., cannot limit computation to just a subset of rules that are of interest to the user). In effect, this model functions as a black box, admitting little user interaction in between.

To address these problems, in [15], we proposed a 2-phase architecture that opens up the black box, and introduced the paradigm of *constrained mining queries*, which together support constraint-based, human-centered exploratory mining of various kinds of rules, including associations. The foundation for the first phase of the architecture is a rich set of constraint constructs, including domain, class, and SQL-style aggregate constraints, which enable users to specify what kind of mined rules are to be computed. The core part in processing constrained mining queries is computing frequent sets that satisfy the specified constraints. This leads to the notion of *constrained frequent set queries* (CFQ). For-

mally, a CFQ<sup>1</sup> is a query of the form:  $\{(S, T) \mid \mathcal{C}\}$ , where  $S, T$  are set variables, and  $\mathcal{C}$  is a set of constraints imposed on  $S, T$ , including the usual frequency constraints. The answer to the CFQ consists of all pairs of frequent sets  $(S, T)$  satisfying  $\mathcal{C}$ . The primary reason why (constrained) frequent sets are chosen to be the intermediate results for the first phase of the computation is that frequent sets represent a common denominator for many kinds of rules of the form  $S \Rightarrow T$ , where  $\Rightarrow$  can mean association rules, correlations, etc. Furthermore, the computation cost of finding (constrained) frequent sets far dominates the cost of forming the final rules (which is done in the second phase of our architecture [15]). Thus, optimizing the computation of (constrained) frequent sets is critical to the success of a system that supports ad-hoc mining.

We illustrate CFQs using the market-basket domain. Apart from the transaction database `trans(TID, Itemset)`, suppose there is auxiliary information stored in the relation `itemInfo(Item, Type, Price)`, which gives the type and price of each item. The CFQ:

$$\{(S, T) \mid S \subset \text{Item} \ \& \ T \subset \text{Item} \ \& \ \text{freq}(S) \ \& \ \text{freq}(T) \ \& \ \text{sum}(S.\text{Price}) \leq 100 \ \& \ \text{avg}(T.\text{Price}) \geq 200\}$$

intends to find pairs of frequent itemsets  $(S, T)$ , where  $S$  has a total price no more than \$100 and  $T$  has an average price no less than \$200. Subsequently, such pairs may be used to compute rules of the form  $S \Rightarrow T$ , suggesting that the purchase of cheaper items “leads to” the purchase of more expensive ones. Here,  $\text{freq}(S)$  says that itemset  $S$  has a support above the user specified threshold. In the sequel, we will drop constraints of the form  $S \subset \text{Item}$  or  $\text{freq}(S)$  and will assume them implicitly. Constraints such as  $\text{sum}(S.\text{Price}) \leq 100$  and  $\text{avg}(T.\text{Price}) \geq 200$  are called *1-var constraints*, because each constraint involves one set variable, and one side of the constraint is a constant. 1-var constraints are useful in conditioning the antecedent and/or consequent separately. The CFQ:

$$\{(S, T) \mid \text{sum}(S.\text{Price}) \leq \text{avg}(T.\text{Price})\}$$

is different from the first query. It uses a *2-var constraint* – involving two set variables. 2-var constraints are useful in constraining the antecedent and consequent jointly.

The main technical results reported in [15] are pruning optimizations – for 1-var constraints – that guarantee a level of performance that is commensurate with the selectivities of the constraints in the user specified query. Those optimizations are based on two key properties of 1-var constraints, namely *anti-monotonicity* and *succinctness*. These properties are exploited in an algorithm called CAP, by pushing the constraints deeply in an Apriori-style levelwise algorithm. We showed in [15] that CAP is effective in bringing about a very significant speedup (e.g., up to 80 times faster) compared with the naive extension to the Apriori algorithm, Apriori<sup>+</sup>, which finds all frequent sets first and then checks them for constraint satisfaction.

As the examples in Section 2 will show, 2-var constraints are natural, ubiquitous, and indispensable in constraining the consequent and antecedent jointly. However, CAP can

only optimize 1-var constraints, and its treatment of 2-var constraints is no smarter than the naive algorithm. This is the subject of this paper. The key question here is: *If there are pruning optimizations that are so effective for 1-var constraints, could there be optimizations as effective for 2-var constraints?* As a preview, this paper provides the following answers to this question:

1. Many association mining algorithms (e.g., the Apriori algorithm and its variants) depend critically on some kind of monotonicity property for their efficiency. The first contribution of this paper is a *negative*, but rather important, result – few 2-var constraints are monotone (or anti-monotone). This reveals the reality that developing pruning optimization for 2-var constraints is a difficult problem, and monotonicity is not the answer this time.
2. Anti-monotonicity and succinctness play a substantial role in optimizing 1-var constraints. Unfortunately, (anti-)monotonicity does not work any more and succinctness does not apply to 2-var constraints directly. To this end, the second contribution of this paper is the concept of *quasi-succinctness* for 2-var constraints. Given a 2-var constraint  $C(S, T)$ , we reduce it to two 1-var succinct constraints of the form  $C_1(S, qc_1)$  and  $C_2(T, qc_2)$ , where  $qc_1, qc_2$  are constants, not given in the query, but can be very efficiently computed. A key technical result is a complete characterization of the class of all quasi-succinct constraints allowed in the CFQ language. Experimental results will show that the speedup achievable for 2-var quasi-succinct constraints is comparable to that achieved for 1-var succinct constraints in [15], while incurring minimal additional overhead.
3. While quasi-succinctness is effective in the optimization of domain and class constraints and aggregation constraints involving  $\min()$  and  $\max()$ , it does not handle 2-var constraints involving  $\text{sum}()$  and/or  $\text{avg}()$ . The third contribution of this paper is two-fold. First, given such a non-quasi-succinct constraint, we show how we can induce a weaker 2-var constraint that *is* quasi-succinct, and can therefore be exploited in optimization as before. Second, because the optimization effected by the weaker induced constraints may be inadequate for some constraint combinations, we develop a heuristic iterative pruning algorithm for those situations. The algorithm is based on a combinatorial analysis of the question: *given all the frequent sets of size  $k$  for some  $k \geq 2$ , what is a good upper bound on the size of the largest frequent set?* Even though CFQs with  $\text{sum}()$  and  $\text{avg}()$  constraints are the hardest to optimize, experimental results will show that the proposed heuristics are effective.
4. The last contribution of this paper is the development of a query optimizer for CFQs. To measure the quality of the computation strategies generated by the optimizer, we propose the notion of *ccc-optimality*. This notion captures the intuition that the effort spent by a strategy in invoking the two fundamental operations – support *counting* and *constraint checking* – should be minimized. We will establish that for a large class of constraints, the query optimizer generates strategies that are ccc-optimal.

<sup>1</sup>In [15], CFQs were called constrained association queries (CAQs). As explained here, CFQs represent a more accurate description of the computation than CAQs.

Section 2 gives more examples of CFQs with 2-var constraints, and summarizes the concepts of anti-monotonicity and succinctness for 1-var constraints. Section 3 introduces and examines anti-monotonicity for 2-var constraints. Section 4 introduces and analyzes quasi-succinctness, and develops pruning optimizations for such constraints. Section 5 develops pruning optimizations for non-quasi-succinct constraints. Section 6 introduces ccc-optimality and presents a query optimizer that generates ccc-optimal strategies for a large class of constraints. Section 7 presents experimental results demonstrating the effectiveness of the optimizations. Section 8 discusses open research problems. For lack of space, the reader is referred to [13] for complete details of the proofs.

## 2 Background

Readers familiar with [15] can skip this section. A CFQ is a query of the form  $\{(S, T) \mid \mathcal{C}\}$ , where  $\mathcal{C}$  is a conjunction of domain, class, and aggregation constraints. For our examples below, we assume the transaction database `trans` (TID, Itemset) with auxiliary information in `itemInfo` (Item, Type, Price). The CFQ

$$\{(S, T) \mid \text{count}(S.\text{Type}) = 1 \ \& \ \text{count}(T.\text{Type}) = 1 \ \& \ S.\text{Type} \neq T.\text{Type}\}$$

asks for pairs of frequent sets containing items of different types (but each set, on its own, containing items of the same type, e.g.,  $\text{count}(S.\text{Type}) = 1$ ). Similarly, the CFQ

$$\{(S, T) \mid S.\text{Type} \cap T.\text{Type} = \emptyset\}$$

asks for frequent itemsets whose associated type sets are disjoint. The CFQ

$$\{(S, T) \mid S.\text{Type} = \{\text{Snacks}\} \ \& \ T.\text{Type} = \{\text{Beers}\} \ \& \ \max(S.\text{Price}) \leq \min(T.\text{Price})\}$$

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

**Definition 1 (1-var anti-monotonicity)** A 1-var constraint  $C$  is *anti-monotone* iff for any set  $S$ :  $S$  does not satisfy  $C \implies \forall S' \supseteq S$ ,  $S'$  does not satisfy  $C$ .

When a 1-var constraint is anti-monotone, it can be optimized in exactly the same way as the frequency constraints are optimized in the well-known Apriori algorithm. A key result in [15] is the characterization of all 1-var anti-monotone constraints, among those allowed in the CFQ language. For any 1-var constraint  $C$ , its *solution space*  $\text{SAT}_C(\text{Item})$  is the set consisting of all the subsets of `Item` that satisfy  $C$ . We refer to elements of  $\text{SAT}_C(\text{Item})$  as *valid sets* w.r.t.  $C$ .

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

The key property of a succinct 1-var constraint  $C$  is that its solution space can be expressed using a succinct description,

which yields a member generating function that generates exactly the solution space  $\text{SAT}_C(\text{Item})$  of  $C$ . In this manner, a succinct constraint can simply operate in a generate-only environment – and need not be in a generate-and-test environment. This leads to significant speedup (e.g., 10 times faster). The following lemma from [15] is important to the subject matter of this paper.

**Lemma 1** 1-var domain, class, and aggregation constraints involving only  $\min()$  and/or  $\max()$  are succinct; 1-var constraints involving  $\text{sum}()$  and/or  $\text{avg}()$  are not. ■

## 3 Anti-monotonicity for 2-var Constraints

Many association mining techniques depend critically on some kind of monotonicity property for efficiency. Given how successful anti-monotone 1-var constraints are in pruning, it is natural to try to imitate the same success in pruning 2-var constraints. There is, however, a huge complication. A 1-var constraint  $C(S)$ , by definition, only has one variable  $S$ , and has one side of the constraint constant. Because this side of the constraint never changes, there is a fixed target for the pruning of  $S$  to take effect. In contrast, a 2-var constraint  $C(S, T)$  has two variables, representing two “degrees of freedom.” Pruning  $S$  is complicated by the fact that  $T$  may change, and vice versa for pruning  $T$ . The following analysis will confirm this observation.

To begin, we need to formalize the notion of the solution space  $\text{SAT}_C$  of a 2-var constraint  $C(S, T)$ . Throughout this paper, for simplicity and concreteness, we assume that  $S$  ranges over the set of items, i.e.  $S \subset \text{Item}$ , and that  $T$  ranges over some domain `Dom`. But for the sake of generality, we will refer to instances of variable  $S$  (resp., variable  $T$ ) as *S-sets* (resp., *T-sets*). Obviously, the definitions are applicable if both variables range over the same domain, i.e.  $\text{Dom} \equiv \text{Item}$ . But assuming that the two variables come from different domains makes it clearer whether  $S$  or  $T$  is being discussed.<sup>2</sup> More importantly, this shows the generality of the framework in allowing two different domains to interact in the same constraint. Furthermore, we assume that there is an attribute  $A$  of elements of `Item` and an attribute  $B$  of elements of `Dom`, such that  $S.A$  and  $T.B$  are in the same domain, to facilitate the interaction of the two domains. In general,  $A$  and/or  $B$  could be absent. For example, if  $T$  ranges over the `Type` domain, then we can speak of a constraint with  $S.\text{Type}$  and  $T$ , such as  $S.\text{Type} \subseteq T$ .

With  $S$  and  $T$  defined as above, the solution space of a 2-var constraint  $C(S, T)$  is given by:

$$\text{SAT}_C(\text{Item}, \text{Dom}) = \{(S_0, T_0) \mid S_0 \subset \text{Item} \ \& \ T_0 \subset \text{Dom} \ \& \ (S_0, T_0) \text{ satisfies } C\}$$

In the sequel, we refer to these  $(S_0, T_0)$  pairs that together satisfy  $C$  as the *valid pairs* w.r.t.  $C$ . For the subject matter discussed later on, we often consider only one variable at a time. This leads to the following definition.

**Definition 3 (Valid S-sets)** For a given 2-var constraint  $C$ , the set of all *valid S-sets* w.r.t.  $C$  is:  $\text{SAT}_C^S(\text{Item}) = \{S_0 \mid \exists T_0 : \text{freq}(T_0) \ \& \ (S_0, T_0) \in \text{SAT}_C(\text{Item}, \text{Dom})\}$ .

<sup>2</sup>It is also possible that even though  $S$  and  $T$  range over the same domain, their associated 1-var constraints may ultimately force them to different parts of the domain. For example, we could have a CFQ with  $\min(S.\text{Price}) \leq 100 \ \& \ \min(T.\text{Price}) \geq 200$  or another CFQ with  $\min(S.\text{Price}) \geq 100 \ \& \ T.\text{Type} = \{\text{Snacks}\}$ .

The set of all valid  $T$ -sets w.r.t.  $C$  can be defined similarly. Note that in the earlier definition of  $\text{SAT}_C(\text{Item}, \text{Dom})$ , the valid pairs  $(S_0, T_0)$  need not be frequent. But in Definition 3, a valid  $S$ -set is one-sided in its use of frequency constraints, since the  $S$ -set need not be frequent. The reason for this asymmetry will become clear in the next section. A valid  $S$ -set that is also frequent, is referred to as a *frequent valid  $S$ -set*.

The spirit of pruning boils down to a smart computation of the set of all frequent, valid pairs that does not require an exhaustive enumeration of all possibilities. In particular, for anti-monotonicity, the hope is that if there is an  $S$ -set  $S_0$  that does not satisfy the constraint in conjunction with all  $T$ -sets examined so far, then all supersets of  $S_0$  cannot possibly satisfy the constraint, and hence, can be safely discarded, regardless of what lies ahead in future computation. This motivates the following definition of anti-monotonicity for 2-var constraints. In the definition, we use the notation  $\text{SAT}_{C,j}^S(\text{Item})$  to denote the set of solutions  $S_0$  related to some frequent  $T$ -set  $T_0$  of size  $\leq j$ , i.e.  $\text{SAT}_{C,j}^S(\text{Item}) = \{S_0 \mid \exists T_0 : \text{freq}(T_0) \ \& \ |T_0| \leq j \ \& \ (S_0, T_0) \in \text{SAT}_C(\text{Item}, \text{Dom})\}$ . By Definition 3,  $\text{SAT}_C^S(\text{Item})$  is identical to  $\bigcup_j \text{SAT}_{C,j}^S(\text{Item})$ .

**Definition 4 (2-var anti-monotonicity)** A 2-var constraint  $C(S, T)$  is *anti-monotone* with respect to  $S$  iff for any  $S$ -set  $S_0$  such that for some integer  $j$ , the pair  $(S_0, T)$  violates  $C$ , for all frequent  $T$ -sets  $T$  of size  $\leq j$ , it is the case that for all supersets  $S'$  of  $S_0$ , the pair  $(S', T')$  violates  $C$ , for all frequent  $T$ -sets  $T'$  of any size, i.e.  $S_0 \notin \text{SAT}_{C,j}^S(\text{Item}) \implies \forall S' \supseteq S_0, S' \notin \text{SAT}_C^S(\text{Item})$ .

Anti-monotonicity w.r.t.  $T$  can be similarly defined. For example,  $S.A \cap T.B = \emptyset$  is an anti-monotone 2-var constraint w.r.t. both  $S$  and  $T$ . Consider a set  $S_0$  such that it is not in  $\text{SAT}_{C,j}^S(\text{Item})$ . This implies  $S_0.A \cap T.B \neq \emptyset$  for all  $T$ -sets of size  $\leq j$ . It is obvious that this violation relationship is preserved when  $S_0$  grows bigger and/or  $T$  grows bigger. Anti-monotonicity w.r.t.  $T$  has an identical argument. Another example of an anti-monotone 2-var constraint is  $\max(S.A) \leq \min(T.B)$ .

Anti-monotone 2-var constraints can lead to effective pruning. Once  $S_0$  is verified not to be in  $\text{SAT}_{C,j}^S(\text{Item})$  for some  $j$  (e.g.,  $j = 1$ ), all its supersets can be removed from consideration. As an effective pruning mechanism for 2-var constraints, this is not the problem with anti-monotonicity. The problem with anti-monotonicity is that there are very few 2-var constraints that are anti-monotone. This statement is based on a detailed analysis we have conducted, from which we have identified the class of all 2-var constraints that are anti-monotone. For space limitations, we do not provide an exhaustive list of combinations and only summarize in Figure 1 the results for a representative subset of 2-var constraints. The second column of the table in Figure 1 identifies which 2-var constraints are anti-monotone. (The third column does the same for quasi-succinctness, which will be discussed in the next section.) Among the domain and class constraints,  $S.A \cap T.B = \emptyset$  is the only anti-monotone 2-var constraint. Among the constraints involving  $\max()$  and/or  $\min()$  shown in the table,  $\max(S.A) \leq \min(T.B)$  is the only instance. And none of the constraints involving  $\text{sum}()$  and  $\text{avg}()$  shown in the table is anti-monotone. We have the following result ascertaining the correctness of the table in Figure 1.

2-var Constraint	Anti-Monotone	Quasi-Succinct
$S.A \cap T.B = \emptyset$	yes	yes
$S.A \cap T.B \neq \emptyset$	no	yes
$S.A \subseteq T.B$	no	yes
$S.A \not\subseteq T.B$	no	yes
$S.A = T.B$	no	yes
$\max(S.A) \leq \min(T.B)$	yes	yes
$\min(S.A) \leq \min(T.B)$	no	yes
$\max(S.A) \leq \max(T.B)$	no	yes
$\min(S.A) \leq \max(T.B)$	no	yes
$\text{sum}(S.A) \leq \max(T.B)$	no	no
$\text{sum}(S.A) \leq \text{sum}(T.B)$	no	no
$\text{avg}(S.A) \leq \text{avg}(T.B)$	no	no

Figure 1: Characterization of 2-var Constraints: Anti-Monotonicity and Quasi-Succinctness

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

**Proof Sketch.** We have already argued why  $S.A \cap T.B = \emptyset$  is anti-monotone. Here we only show the proof of one negative case, namely  $C \equiv \min(S.A) \leq \min(T.B)$ . Consider a set  $S_0$  such that it is not in  $\text{SAT}_{C,j}^S(\text{Item})$  for some integer  $j$ . In other words,  $\min(S_0.A) > \min(T.B)$  for all  $T$ -sets of size  $\leq j$ . However, for all supersets  $S'$  of  $S_0$ ,  $\min(S_0.A) \geq \min(S'.A)$ . Thus, it is possible that there may be a  $T_0$  such that  $\min(S'.A) \leq \min(T_0.B)$ , thus putting  $(S', T_0)$  in the solution space. ■

Though not exhaustive, Figure 1 captures the reality that few 2-var constraints are anti-monotone. This is a negative, but important, result, showing the difficulty in optimizing 2-var constraints.

## 4 Quasi-succinctness

The analysis conducted in the previous section reveals that when pruning for a variable in a 2-var constraint, it is important to have the other variable present a fixed target – not one that keeps on changing. This forms the basis for the concept of *quasi-succinctness* to be introduced below. Intuitively, a 2-var constraint  $C(S, T)$  is *quasi-succinct* if it can be reduced to two 1-var succinct constraints of the form  $C_1(S, qc_s)$  and  $C_2(T, qc_t)$ , where  $qc_s, qc_t$  are constants such that the set of all valid  $S$ -sets and the set of all valid  $T$ -sets are preserved under the reduction. The motivation for such a definition is that we would like to decouple the dependency or the constraint binding the two variables together so that pruning for  $S$  and  $T$  can occur independently, and as soon as possible. In this section, we first present a detailed analysis of one particular constraint so as to introduce the various concepts associated with quasi-succinctness. Then we simply summarize our quasi-succinctness results for many other constraints.

### 4.1 The Non-overlapping Constraint: a Case Study

As a concrete example, we consider the constraint  $C(S, T) \equiv S.A \cap T.B = \emptyset$ . Throughout this paper, we use the notation  $L_j^S$  to denote the set of all elements contained in any frequent  $S$ -set of size  $j$ , i.e.  $L_j^S = \{e \mid \exists S : S \subset \text{Item} \ \& \ \text{freq}(S) \ \& \ |S| = j \ \& \ e \in S\}$ . Similarly,  $L_j^T$  denotes the set of all elements contained in any frequent  $T$ -set of size  $j$ . As usual, the notation  $L_j^S.A$  denotes  $\{e.A \mid e \in L_j^S\}$ .

Given the constraint  $S.A \cap T.B = \emptyset$ , the goal is to find 1-var succinct constraints for pruning candidate  $S$ - and  $T$ -sets. The following lemma shows that for a candidate set  $CS \subset \text{Item}$  to be a valid  $S$ -set w.r.t.  $S.A \cap T.B = \emptyset$ , it is necessary that  $CS.A$  does *not* contain all elements in  $L_1^T.B$ .

**Lemma 2** Let  $CS$  be a candidate  $S$ -set, i.e.  $CS \subset \text{Item}$ . Then:

$$\exists \text{ a frequent } T\text{-set } T \text{ such that } CS.A \cap T.B = \emptyset \implies CS.A \not\supseteq L_1^T.B.$$

**Proof Sketch.** Suppose  $CS.A \supseteq L_1^T.B$ . Let  $T$  be a frequent  $T$ -set of any size. By the definition of frequent sets, we have  $T \subseteq L_1^T$ , and thus  $T.B \subseteq CS.A$ . This implies  $CS.A \cap T.B \neq \emptyset$ . ■

The 1-var constraint,  $C_1(S) \equiv CS.A \not\supseteq L_1^T.B$ , can be regarded as a *pruning condition* for candidate  $S$ -sets. The above lemma gives a *sound* pruning condition for candidate  $S$ -sets. A pruning condition  $C_1$  is sound w.r.t. the original 2-var constraint  $C$ , if it does not prune away any valid  $S$ -set, i.e.,  $CS \in \text{SAT}_C^S(\text{Item}) \implies CS \in \text{SAT}_{C_1}^S(\text{Item})$ . Conversely, a pruning condition  $C_1$  for  $S$ -sets is *tight*<sup>3</sup> w.r.t. the original 2-var constraint  $C$ , if it prunes away every  $S$ -set that is not valid, i.e.,  $CS \in \text{SAT}_{C_1}^S(\text{Item}) \implies CS \in \text{SAT}_C^S(\text{Item})$ . The following lemma shows that, apart from being sound, the condition  $CS.A \not\supseteq L_1^T.B$  is also a tight pruning condition for candidate  $S$ -sets.

**Lemma 3** Let  $CS$  be a candidate  $S$ -set, i.e.  $CS \subset \text{Item}$ . Then:

$$CS.A \not\supseteq L_1^T.B \implies \exists \text{ a frequent } T\text{-set } T \text{ such that } CS.A \cap T.B = \emptyset.$$

**Proof Sketch.**  $CS.A \not\supseteq L_1^T.B$  implies that there exists an element  $t \in L_1^T$  such that  $t.B \not\subseteq CS.A$ . But by definition of  $L_1^T$ , the set  $\{t\}$  is frequent. Thus, there exists a frequent set – namely,  $\{t\}$  – such that  $CS.A \cap \{t\}.B = \emptyset$ . ■

Recall that in Definition 3, valid  $S$ -sets are defined based on frequent  $T$ -sets. If they were defined without requiring  $T$ -sets to be frequent, the above two lemmas would not be true, and the given constraint would not be a sound and tight pruning condition. This explains why in Definition 3, there is the one-sided use of the frequency constraints. The following corollary gives a sound and tight condition for pruning candidate sets for variable  $T$ .

**Corollary 1** Let  $CT$  be a candidate  $T$ -set, i.e.  $CT \subset \text{Dom}$ . Then:  $\exists \text{ a frequent } S\text{-set } S \text{ such that } S.A \cap CT.B = \emptyset \iff CT.B \not\supseteq L_1^S.A$ . ■

**Definition 5 (Quasi-succinctness)** Constraint  $C(S, T)$  is *quasi-succinct* if it can be reduced to two 1-var constraints  $C_1(S), C_2(T)$  such that: (i)  $C_1$ , involving only the variable  $S$ , is succinct, and is a sound and tight pruning condition for candidate  $S$ -sets; and (ii)  $C_2$ , involving only  $T$ , is succinct, and is a sound and tight pruning condition for candidate  $T$ -sets.

The above two lemmas and corollary show that the constraint  $C \equiv S.A \cap T.B = \emptyset$  is quasi-succinct, because as summarized in Lemma 1, the reduced constraints  $C_1(S) \equiv S.A \not\supseteq L_1^T.B$  and  $C_2(T) \equiv T.B \not\supseteq L_1^S.A$  are succinct 1-var constraints. This is great news from a computational

<sup>3</sup>Note that soundness and tightness are defined w.r.t. pruning – not satisfaction – which explains the direction of the implications.

2-var constraint $C$	sound & tight $C_1(S)$	sound & tight $C_2(T)$
$S.A \cap T.B = \emptyset$	$CS.A \not\supseteq L_1^T.B$	$CT.B \not\supseteq L_1^S.A$
$S.A \cap T.B \neq \emptyset$	$CS.A \cap L_1^T.B \neq \emptyset$	$CT.B \cap L_1^S.A \neq \emptyset$
$S.A \subseteq T.B$	$CS.A \subseteq L_1^T.B$	$L_1^S.A \cap CT.B \neq \emptyset$
$S.A \not\subseteq T.B$	$(CS \neq \emptyset)$	$L_1^S.A \not\subseteq CT.B$
$S.A = T.B$	$CS.A \subseteq L_1^T.B$	$CT.B \subseteq L_1^S.A$

Figure 2: Quasi-succinctness: Reduction of 2-var Domain Constraints

standpoint. This is because succinct 1-var constraints can operate in a generate-only environment, thus avoiding a generate-and-test environment. Consequently, significant speedup can be achieved. Now, thanks to quasi-succinctness, the speedup that can be achieved for 1-var succinct constraints is directly applicable to optimizing 2-var quasi-succinct constraints. Furthermore, in  $C_1(S)$  and  $C_2(T)$  above, the constants in the constraints are the sets  $L_1^T.B$  and  $L_1^S.A$  respectively. A key point here is that these sets  $L_1^S$  and  $L_1^T$  are computed in any event for frequency verification purposes. Thus, the de-coupling process in quasi-succinctness requires little extra cost. Last but not least, in a customary, levelwise computational framework, the subscript 1 in both  $L_1^S$  and  $L_1^T$  implies that immediately after the first iteration of counting, the 2-var constraint can be de-coupled to effect separate pruning.

## 4.2 Other Domain Constraints

Based on the notion of quasi-succinctness illustrated so far, we have conducted a detailed analysis of 2-var constraints. Column 3 of the table in Figure 1 gives a complete characterization of a representative subset of quasi-succinct constraints among those allowed in our CFQ language. Basically, all domain 2-var constraints are quasi-succinct. We will comment on the other 2-var constraints shortly. For the domain constraints shown in Figure 1, the table in Figure 2 shows their corresponding 1-var succinct constraints  $C_1(S)$  and  $C_2(T)$ . We have the following formal result to ascertain the correctness of the entries in Figure 2.

**Theorem 2** For each 2-var constraint  $C$  listed in the table in Figure 2, the following holds:

- $C_1(S)$  is a succinct, sound and tight pruning condition for candidate  $S$ -sets; and
- $C_2(T)$  is a succinct, sound and tight pruning condition for candidate  $T$ -sets. ■

The first entry in the table is proved by Lemmas 2, 3 and Corollary 1. For lack of space, we do not include other proofs here. But concerning the other entries in the table, we make one observation: some of the 1-var constraints shown in the table actually have less pruning power than others. An extreme example is  $S.A \neq T.B$ , in which case the corresponding 1-var constraint for  $S$  is  $CS \neq \emptyset$ , which has virtually no pruning power.

## 4.3 Aggregation Constraints Involving Only $\min()$ and $\max()$

Next we turn our attention to 2-var aggregate constraints. First, we focus on 2-var constraints of the form:  $\text{agg}_1(S.A) \theta \text{agg}_2(T.B)$ , where  $\text{agg}_1, \text{agg}_2$  are either  $\min()$  or  $\max()$ ,  $\theta$

2-var constraint $C$	sound & tight $C_1(S)$	sound & tight $C_2(T)$
$\min(S.A) \leq \min(T.B)$	$\min(CS.A) \leq \max(L_1^T.B)$	$\min(CT.B) \geq \min(L_1^S.A)$
$\min(S.A) \leq \max(T.B)$	$\min(CS.A) \leq \max(L_1^T.B)$	$\max(CT.B) \geq \min(L_1^S.A)$
$\max(S.A) \leq \min(T.B)$	$\max(CS.A) \leq \max(L_1^T.B)$	$\min(CT.B) \geq \min(L_1^S.A)$
$\max(S.A) \leq \max(T.B)$	$\max(CS.A) \leq \max(L_1^T.B)$	$\max(CT.B) \geq \min(L_1^S.A)$

Figure 3: Quasi-succinctness: Reduction of  $\min()$  and  $\max()$  Constraints

is one of  $=, \leq, \geq$ , and  $S.A$  and  $T.B$  are in the same domain. Again because there are many combinations, we only summarize a few cases in Figures 1 and 3. Other cases not shown can be handled similarly [13]. In terms of formal results, as shown in the theorem below, in each case, the accompanying 1-var constraints are guaranteed to be succinct, sound and tight pruning conditions. For lack of space, we only show a proof sketch for one 2-var constraint,  $C \equiv \max(S.A) \leq \max(T.B)$ . It will become obvious later why we pick this constraint for elaboration.

**Theorem 3** For each 2-var constraint  $C(S, T)$  listed in Figure 3, the following holds:

- $C_1(S)$  is a succinct, sound and tight pruning condition for candidate  $S$ -sets; and
- $C_2(T)$  is a succinct, sound and tight pruning condition for candidate  $T$ -sets.

**Proof Sketch.** Consider  $C \equiv \max(S.A) \leq \max(T.B)$ . The following is to show that: (i)  $CS$  is a valid  $S$ -set w.r.t.  $C$  iff  $CS$  satisfies the constraint  $\max(CS.A) \leq \max(L_1^T.B)$ ; and (ii)  $CT$  is a valid  $T$ -set w.r.t.  $C$  iff  $CT$  satisfies  $\max(CT.B) \geq \min(L_1^S.A)$ .

(For  $CS$ :) Suppose  $CS$  satisfies  $\max(CS.A) > \max(L_1^T.B)$ . For any frequent  $T$ -set  $T$ , it is the case that  $\max(L_1^T.B) \geq \max(T.B)$ , because  $T \subseteq L_1^T$ . Thus, there will never be a frequent  $T$ -set  $T_0$  such that the pair  $(CS, T_0)$  satisfies constraint  $C$ , so  $CS$  cannot be valid. Now suppose  $CS$  satisfies  $\max(CS.A) \leq \max(L_1^T.B)$ . Then there exists an element  $t \in L_1^T$  such that  $\max(CS.A) \leq t.B$ . Now consider the set  $\{t\}$ . It is frequent, and the pair  $(CS, \{t\})$  satisfies  $C$ , implying  $CS$  is valid.

(For  $CT$ :) Suppose  $CT$  satisfies  $\max(CT.B) < \min(L_1^S.A)$ . Then as argued above, it is clear that for any frequent  $S$ -set  $S$ ,  $\min(L_1^S.A) \leq \max(S.A)$ . Thus, there cannot be a frequent  $S$ -set  $S_0$  such that the pair  $(S_0, CT)$  satisfies constraint  $C$ . Now suppose  $CT$  satisfies  $\max(CT.B) \geq \min(L_1^S.A)$ . Then there exists an element  $s \in L_1^S$  such that  $\max(CT.B) \geq s.A$ . This makes  $CT$  a valid  $T$ -set. ■

There are two reasons why we choose to include the above proof here. First, this serves as a concrete example of the arguments showing the soundness and tightness of the pruning conditions given in Figure 3. Second, the proof explains many interesting regularities that exist among the conditions given in the figure. For instance, the succinct constraint  $C_1(S)$  is identical in the third and fourth rows of the table. The observation here is that the proof for  $CS$  given above works for either of the two 2-var constraints

$\max(S.A) \leq \min(T.B)$  or  $\max(S.A) \leq \max(T.B)$ . Similarly, notice the similarity between the succinct constraints  $C_2(T)$  for the same two constraints. The term  $\min(L_1^S.A)$  appears in both succinct constraints, and  $\min(T.B)$  in the 2-var constraint corresponds to  $\min(CT.B)$  in the succinct 1-var constraint, and vice versa for  $\max(T.B)$ . Again the proof for  $CT$  above explains why.

## 5 Optimizing 2-var Constraints Involving $\text{sum}()$ and $\text{avg}()$

In the previous section, we have analyzed quasi-succinct constraints. Next we turn to non-quasi-succinct constraints. These are constraints involving  $\text{sum}()$  and  $\text{avg}()$ . Specifically, we develop a two-pronged approach for optimizing non-quasi-succinct constraints. First, we show how such a constraint can induce weaker 2-var constraints that are quasi-succinct, thereby making use of the results presented in the previous section. Second, because for some constraint combinations, the induced constraints may not always yield adequate pruning by themselves, we develop an iterative heuristic pruning algorithm of a different flavor.

### 5.1 Inducing Weaker Quasi-succinct Constraints

To illustrate, consider the constraint  $C \equiv \text{sum}(S.A) \leq \max(T.B)$ . Constraint  $C$  implies the weaker constraint  $C' \equiv \max(S.A) \leq \max(T.B)$ , in the sense that  $CS \in \text{SAT}_C^S(\text{Item}) \implies CS \in \text{SAT}_{C'}^S(\text{Item})$ , and similarly for  $CT$ . Intuitively, for any candidate  $S$ -sets  $CS$  violating  $C'$ ,  $CS$  must also violate  $C$ . (The results in this section assume that the domains of  $A$  and  $B$  are non-negative.) We know from Section 4 that  $C'$  is quasi-succinct. Thus, we can use the 1-var succinct constraints  $C_1(S)$  and  $C_2(T)$  (see the table in Figure 3) as pruning conditions for  $C$ . Notice that even though  $C_1(S)$  and  $C_2(T)$  are sound and tight w.r.t.  $C'$ , they are only sound pruning conditions for candidate  $S$ -sets and  $T$ -sets w.r.t.  $C$ . Because the pruning is not tight, when eventually the valid pairs w.r.t.  $C$  are computed, an additional verification against  $C$  must be performed.

Based on the idea of inducing weaker constraints, the table in Figure 4 shows the sound pruning conditions  $C_1(S)$  and  $C_2(T)$  w.r.t.  $C$  for a representative subset of aggregate constraints involving  $\text{sum}()$  and/or  $\text{avg}()$ . In general, with  $\text{agg}()$  denoting any aggregation allowed in our language, (i)  $C \equiv \text{avg}() \leq \text{agg}()$  induces  $C' \equiv \min() \leq \text{agg}()$ ; (ii)  $C \equiv \text{sum}() \leq \text{agg}()$  induces  $C' \equiv \max() \leq \text{agg}()$ ; and (iii)  $C \equiv \text{agg}() \leq \text{avg}()$  induces  $C' \equiv \text{agg}() \leq \max()$ . We have the following lemma.

**Lemma 4** For each 2-var aggregate constraint  $C$  given in the table of Figure 4, we have:

- $C_1(S)$  is a succinct and sound pruning condition for candidate  $S$ -sets; and
- $C_2(T)$  is a succinct and sound pruning condition for candidate  $T$ -sets. ■

Induced weaker constraints can be quite effective in pruning for many cases. But sometimes they may be too “loose”, particularly for constraints involving only  $\text{sum}()$  and/or  $\text{avg}()$ . A perfect example is the constraint  $C \equiv \text{sum}(S.A) \leq \text{sum}(T.B)$ . It is not difficult to show that a sound pruning

2-var constraint $C$	induced weaker constraint $C'$	sound $C_1(S)$	sound $C_2(T)$
$avg(S.A) \leq min(T.B)$	$min(S.A) \leq min(T.B)$	$min(CS.A) \leq max(L_1^T.B)$	$min(CT.B) \geq min(L_1^S.A)$
$sum(S.A) \leq max(T.B)$	$max(S.A) \leq max(T.B)$	$max(CS.A) \leq max(L_1^T.B)$	$max(CT.B) \geq min(L_1^S.A)$
$avg(S.A) \leq avg(T.B)$	$min(S.A) \leq max(T.B)$	$min(CS.A) \leq max(L_1^T.B)$	$max(CT.B) \geq min(L_1^S.A)$

Figure 4: Induced Weaker Constraints for Constraints involving  $sum()$  and/or  $avg()$

1. Set  $N_i^k$  to be the number of frequent sets of size  $k$  containing  $t_i$ .
2. Set  $J_i^k$  to be the largest  $j$  with:  $N_i^k \geq \binom{k+j-1}{k-1}$  (1)
3. Set  $J_{max}^k$  to be  $\max\{J_i^k \mid 1 \leq i \leq m\}$ .

Figure 5: Computing an Upper Bound on Largest Frequent  $T$ -set,  $J_{max}^k$ .

condition  $C_1(S)$  is  $sum(CS.A) \leq sum(L_1^T.B)$ . But the constant  $sum(L_1^T.B)$  can be too large. The following numerical example, which we will reuse later, illustrates this situation. Suppose  $L_1^T = \{t_1, \dots, t_{100}\}$ . Suppose  $t_i.B = i$  for all  $1 \leq i \leq 100$ . Then the pruning condition  $sum(CS.A) \leq sum(L_1^T.B)$  yields  $sum(CS.A) \leq (1 + \dots + 100) = 5050$ . Below we develop another pruning technique primarily for constraints involving only  $sum()$  and/or  $avg()$ .

## 5.2 Heuristic Iterative Pruning with $J_{max}^k$

The problem we have just seen about pruning  $sum(S.A) \leq sum(T.B)$  raises the following question: if we cannot produce tight pruning conditions with one constraint and one constant, can we produce a series of constraints with increasingly stronger pruning power? Let us first observe that between  $S$  and  $T$ , the pruning condition for  $S$ , having the form  $sum(S.A) \leq V_A$ , is likely to be of more value, because the latter constraint is also anti-monotone (cf: Definition 1), which permits effective optimization. Thus, let us focus our attention on the  $S$  side, in the rest of this section. An obvious – but ineffective – choice for the value  $V_A$  is  $sum(L_1^T.B)$ . Instead, in the following, we show how to generate a series of values  $V_A^2, \dots, V_A^k$  based on the frequent  $T$ -sets of size 2 through  $k$ . To do so, we must first answer the following question: given all the frequent  $T$ -sets of size  $k$  for some  $k \geq 2$ , what is an upper bound on the size of the largest (in the cardinality sense) frequent  $T$ -set?

The procedure given in Figure 5 provides such a bound. Given all the frequent  $T$ -sets of size  $k$ , let  $t_1, \dots, t_m$  be an enumeration of all the elements contained in any frequent  $T$ -set of size  $k$ , i.e., elements in  $L_k^T$ . The intuition behind Equation (1) in Figure 5 is that in order for the element  $t_i$  to appear in at least one frequent  $T$ -set of size  $k+j$ , it must appear in at least  $\binom{k+j-1}{k-1}$  frequent sets of size  $k$ . Thus,  $J_i^k$ , being the maximum of all  $j$ 's satisfying Equation (1), gives an upper bound on the largest frequent  $T$ -set containing  $t_i$ . Hence, given all frequent  $T$ -sets of size  $k$ ,  $J_{max}^k$  is an upper bound on the size of the largest frequent  $T$ -set.

To continue with our earlier numerical example, suppose there are 17 frequent sets of size 4 containing element  $t_1$ , i.e.,  $N_1^4 = 17$ . Then it is not possible to have a frequent set of size 7 containing  $t_1$ , because otherwise, there should

1. For an arbitrary element  $t_i \in L_k^T$ , among all frequent  $T$ -sets of size  $k$  containing  $t_i$ , let the set  $T_i^k$  be the one with the maximum value of  $sum(T.B)$ . Let that sum be  $Sum_i^k$ .
2. Let  $E_i^k$  be the set of all elements of  $L_k^T$  that are not in  $T_i^k$  but co-occurring with  $t_i$  in some frequent set of size  $k$ . Let  $e_1, \dots, e_w$  be an enumeration of all the elements in  $E_i^k$  in descending order of their  $B$ -values, i.e.  $e_1.B \geq \dots \geq e_w.B$ .
3. Set  $MaxSum_i^k$  to be  $Sum_i^k + \sum_{u=1}^{J_{max}^k} e_u.B$ .
4. Set  $V^k$  to be  $\max\{MaxSum_i^k \mid 1 \leq i \leq m\}$ .

Figure 6: Iterative Pruning Using  $J_{max}^k$ .

at least be  $\binom{7-1}{4-1} = 20$  frequent sets of size 4 containing  $t_1$ . In other words, the largest frequent set containing  $t_1$  is of size at most 6, i.e.,  $J_1^4 = 2$ . Depending on the actual distribution of the elements in the 17 frequent sets of size 4, and the frequency counts of the sets, the actual largest frequent set containing  $t_1$  may in fact have size smaller than 6. But the point is that the best estimate we can make from the given information is 6.

The following lemma says that for each element, as we increase our knowledge from knowing all frequent sets of size  $k$  to knowing all frequent sets of size  $k+1$ , we can sharpen our upper bound.

**Lemma 5** For all  $k \geq 2$ , it is necessary that for all  $1 \leq i \leq m$ ,  $J_i^{k+1} < J_i^k$  and  $J_{max}^{k+1} < J_{max}^k$ . ■

Before we return to the discussion of optimizing 2-var constraints involving  $sum()$  and  $avg()$ , we point out that  $J_{max}^k$  can be computed very efficiently. All the quantities  $N_i^k$  can be computed with one pass over all the frequent  $T$ -sets of size  $k$ . These “counters” may best be maintained on-the-fly as the frequent sets are computed. Regarding Steps 2 and 3 in Figure 5, it is easy to see that we can execute Step 3 only once based on the maximum  $N_i^k$  value, instead of solving Equation (1)  $m$  times for each  $i$ ,  $1 \leq i \leq m$ . Thus, the time taken to find  $J_{max}^k$  is negligible.

Recall that our objective is to effect iterative pruning for the constraint  $sum(S.A) \leq sum(T.B)$ , by producing a series of 1-var constraints  $sum(S.A) \leq V^i$ ,  $1 \leq i \leq k$ , where the upper bounds get tighter as  $i$  increases. Figure 6 shows how this series can be produced. To continue with our earlier example, suppose for the element  $t_{100}$ , that the frequent  $T$ -set of size 4 containing  $t_{100}$  that has the maximum  $sum(T.B)$  value, is the set  $\{t_{10}, t_{50}, t_{80}, t_{100}\}$ . This set gives a total sum of  $Sum_{100}^4 = 240$ . (Recall that we are assuming, for simplicity, that  $t_i.B = i$ .) Suppose that  $J_{max}^4$  is 2, and that among the other elements co-occurring with  $t_{100}$

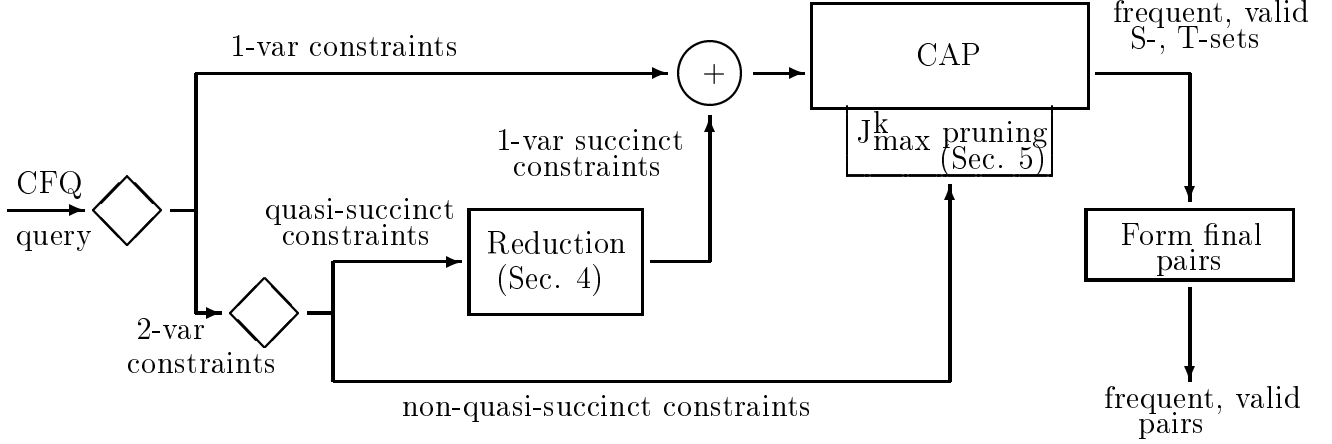


Figure 7: A Schematic Diagram of the CFQ Query Optimizer

in any frequent  $T$ -set of size 4, the elements with the top-2  $B$ -values are  $t_{90}$  and  $t_{70}$ . Then the value of  $MaxSum_{100}^4$  is given by  $240 + 90 + 70 = 400$ . It is easy to see that for any frequent  $T$ -set  $T$  containing  $t_{100}$ ,  $sum(T.B)$  is bounded from above by 400.

**Lemma 6** For any value of  $k \geq 2$ , if  $CS$  is a valid  $S$ -set w.r.t. the constraint  $sum(S.A) \leq sum(T.B)$ , then it necessarily satisfies  $sum(CS.A) \leq V^k$ . ■

**Lemma 7** For all  $k \geq 2$ , it is necessary that  $V^{k+1} \leq V^k$ . ■

In closing this section, we offer three extra comments. First, we have discussed how to use  $J_{max}^k$  to derive a series of  $V^k$  values for  $sum(S.A) \leq V^k$ . Given the constraint  $avg(S.A) \leq agg(T.B)$ , where  $agg()$  is  $sum()$  or  $avg()$ , we can derive a series of  $A^k$  values for  $avg(S.A) \leq A^k$ .

Second, iterative pruning with  $J_{max}^k$  makes sense only when the lattice computations for  $S$  and  $T$  are “dovetailed”, in that computing one level of the lattice for  $S$  is followed by one level of  $T$  and vice versa. An attentive reader would raise the following objection to dovetailing. Suppose the only constraint in a given CFQ is  $sum(S.A) \leq sum(T.B)$ . Then, as argued above, we can expect to do very little pruning on the  $T$  side for this constraint. One reasonable strategy is to compute all frequent  $T$ -sets, find the global maximum  $M = \max\{sum(T.B) \mid freq(T)\}$ , and then use the condition  $sum(S.A) \leq M$  as a pruning condition for candidate  $S$ -sets. Purely from the viewpoint of pruning, this is convincing. However, this argument ignores the I/O cost. In general, dovetailing between the lattices for  $S$  and  $T$  allows for sharing of scans on the transaction database, from which frequency constraints are verified. While it remains an open problem as to what the optimal strategy for computing a CFQ is, counting both CPU and I/O costs, we believe dovetailing is reasonable under many circumstances. In those cases, iterative pruning based on  $J_{max}^k$  is an attractive strategy.

Finally, we note that using induced weaker constraints and iterative pruning with  $J_{max}^k$  are complementary and work at different times. Induced weaker constraints effect pruning

once and for all right after iteration 1 in frequency counting, whereas pruning with  $J_{max}^k$  comes into effect multiple times in subsequent iterations.

## 6 A CFQ Query Optimizer

So far our focus has been on how to optimize different kinds of 2-var constraints on an individual basis. In this section, we tie all these different pieces together into a query optimization framework. Specifically, we present a CFQ query optimizer which given a CFQ, produces an optimized computation strategy for the CFQ, considering both 1-var and 2-var constraints. To evaluate the quality of the strategy produced by the optimizer, we introduce the notion of *ccc-optimality*. This notion seeks to capture the effort spent by a strategy in invoking two fundamental operations – support counting and constraint checking. We show that for a large class of constraints, the strategy is *ccc-optimal*.

### 6.1 A Schematic Diagram of the Optimizer

Figure 7 shows how the CFQ query optimizer operates when presented with a set  $\mathcal{C}$  of constraints. It first separates the 1-var and 2-var constraints, i.e.  $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ . This separation is purely syntactic. The set  $\mathcal{C}_2$  of 2-var constraints is then further divided into two subsets  $\mathcal{C}_{qs}, \mathcal{C}_{nqs}$ , as follows.  $\mathcal{C}_{qs}$  contains every quasi-succinct constraint in  $\mathcal{C}_2$ , while  $\mathcal{C}_{nqs} = \mathcal{C}_2 - \mathcal{C}_{qs}$ . Using the ideas in Section 5.1, from each constraint in  $\mathcal{C}_{nqs}$ , a weaker quasi-succinct constraint is induced and added to the set  $\mathcal{C}_{qs}$ . Then based on the material in Section 4, each quasi-succinct 2-var constraint in  $\mathcal{C}_{qs}$  is reduced to two 1-var succinct constraints. This transforms  $\mathcal{C}_{qs}$  into the corresponding  $\mathcal{C}_{qs,1}$ . The latter is then put together with the set of 1-var constraints from the initial CFQ, i.e.  $\mathcal{C}_{qs,1} \cup \mathcal{C}_1$ . Together these constraints are exploited by the CAP algorithm which provides optimized execution of 1-var constraints [15]. If the set  $\mathcal{C}_{nqs}$  of non-quasi-succinct constraints is not empty, then the iterative pruning strategy developed in Section 5.2 is applied. Specifically, at each level of the lattice computation by CAP (and for that matter, Apriori as well), a candidate set is only counted for frequency verification if it satisfies the constraint induced by  $J_{max}^k$ . This additional filtering

step is depicted in Figure 7 as an add-on to the CAP module, which produces all frequent, valid  $S$ - and  $T$ -sets (cf. Definition 3).<sup>4</sup> Finally, from among these sets, the frequent, valid pairs are formed. This final step is trivial if there is no 2-var constraint in the CFQ to begin with.

## 6.2 Performance Guarantee: CCC-Optimality

The query optimizer outlined in Figure 7 generates a specific strategy for computing a given CFQ. We evaluate below the quality of this strategy. As shown in Figure 7, to find all the frequent, valid pairs  $(S, T)$  for a given CFQ, there are two steps involved: (i) finding all the frequent, valid  $S$ - and  $T$ -sets; and (ii) forming the pairs. The diagram in Figure 7 suggests that the first step requires a lot more computational effort than the second step. Indeed, experimental results indicate that the first step typically requires a total runtime many orders of magnitude higher than what the second step needs. Thus, in the ensuing discussion, we only focus on the performance of the first step.

To measure performance, we consider two cost components: (i) the effort needed for constraint checking, and (ii) that for support counting. The level of granularity of our cost model is such that for constraint checking, we count the number of invocations of the constraint checking operation, and for support counting we count the number of sets for which support is counted. This leads to the notion of *ccc-optimality*, where “ccc” stands for constraint checking and counting. We say that a candidate  $S$ -set  $CS$  is valid w.r.t. a set of constraints  $\mathcal{C}$  if  $CS$  is valid w.r.t. all 1-var and 2-var constraints in  $\mathcal{C}$  in the sense defined in Sections 2 and 3.

**Definition 6 (ccc-optimality)** A computation strategy is *ccc-optimal* for a class of constraints provided for every set  $\mathcal{C}$  of constraints from that class, it satisfies the following conditions:

- (1) the strategy counts for the support of a candidate set  $CS$  iff: all subsets of  $CS$  are frequent, and  $CS$  is valid;
- (2) the strategy invokes the constraint checking operation on a candidate  $S$ -set  $CS$ , only if  $|CS| = 1$ .

The first condition of ccc-optimality guarantees that when a set  $CS$  is counted for its support: (i) all subsets of  $CS$  are frequent, (ii)  $CS$  satisfies all 1-var constraints in  $\mathcal{C}$  involving  $S$ , and (iii) for every 2-var constraint  $C$  in  $\mathcal{C}$  involving  $S$  and  $T$ , there necessarily exists a frequent  $T$ -set  $CT$ , such that  $(CS, CT)$  is a valid pair w.r.t.  $\mathcal{C}$ .<sup>5</sup> At this stage, the frequency constraint is the only remaining requirement that could possibly prevent  $CS$  from becoming a frequent, valid  $S$ -set. Thus, in a bottom-up levelwise computational framework, the first condition, in some sense, represents the minimum number of sets that need to be counted for support verification.

Given a set  $\mathcal{C}$  of constraints, the naive algorithm, called *Apriori*<sup>+</sup> in [15], can compute all frequent, valid sets by

<sup>4</sup>If induced weaker constraints are used for non-quasi-succinct constraints, as described in Section 5.1, the output of CAP may include, in addition to all the valid  $S$ - and  $T$ -sets, some  $S$ - and  $T$ -sets not valid for the original constraints. Those will be discarded when the final step of forming valid pairs is executed.

<sup>5</sup>We stress that we may not, and need not, know what exactly  $CT$  is, as long as we know some such  $CT$  must exist. This is the power of the material on quasi-succinctness in Section 4.

first computing all frequent sets, and then verifying whether these frequent sets satisfy  $\mathcal{C}$ . It is easy to see that for most instances of  $\mathcal{C}$ , *Apriori*<sup>+</sup> is not ccc-optimal because it violates the first condition by counting sets that are invalid. Surprisingly, there are instances of  $\mathcal{C}$  for which *Apriori*<sup>+</sup> is ccc-optimal; we will characterize those instances shortly.

The first condition alone, however, does not necessarily guarantee that the invocation of the constraint checking operation is done a minimal number of times. As a counterexample, consider the following “full-materialized” (FM) strategy. FM first computes all valid sets by generating all possible subsets and verifying each and every one against the set  $\mathcal{C}$  of constraints. Then among all the subsets that satisfy  $\mathcal{C}$ , it counts the support in ascending cardinality. Clearly, FM satisfies the first condition above, as it counts the minimum number of sets for support. Equally clearly, FM leaves much to be desired as it performs constraint checking too many times, indeed  $2^N$  times in the worst case, where  $N$  is the size of the active domain (e.g., the number of items).

This motivates the second condition in Definition 6, namely that the number of invocations of constraint checking is restricted to at most  $N$ . In a bottom-up levelwise computational framework, we believe this is a reasonable lower bound, since it is not clear how one can do with fewer constraint checking invocations in that framework. With this motivation, we contend that ccc-optimality is a very desirable goal for a computation strategy for CFQ to achieve.

**Theorem 4** Algorithm CAP is ccc-optimal for the class of 1-var succinct constraints. ■

Recall from Figure 7 that Algorithm CAP is used to process 1-var constraints. The above theorem states that CAP achieves ccc-optimality for  $\mathcal{C}$  if the set consists of only 1-var succinct constraints. While the details of a proof are given in [13], the general idea is that to any set of 1-var succinct constraints, including those which are not anti-monotone, there is a corresponding function, called the member generating function (MGF) in [15], that can generate exactly those sets that satisfy the constraints. The MGF operates in such a way that the second condition of ccc-optimality is satisfied. Together with the MGF, CAP then guarantees that the first condition of ccc-optimality is also met.

**Corollary 2** The strategy generated by the CFQ query optimizer is ccc-optimal for the class of constraints consisting of 1-var succinct and 2-var quasi-succinct constraints. ■

Recall from Theorems 2 and 3 that each 2-var quasi-succinct constraint can be reduced to two 1-var succinct constraints preserving the valid  $S$ - and  $T$ -sets. These reduced 1-var succinct constraints can be set up appropriately after all the sets in the first levels of the lattices for  $S$  and  $T$  have been counted for their support. This ensures that the second condition of ccc-optimality is satisfied. Then by virtue of Theorem 4, the computation strategy given by the CFQ query optimizer makes use of CAP to guarantee that ccc-optimality is achieved w.r.t. the reduced 1-var succinct constraints. This then, by Definition 6, implies that this strategy is ccc-optimal w.r.t. the 2-var quasi-succinct constraints as well.

In general, *Apriori*<sup>+</sup> is not ccc-optimal for 1-var succinct constraints, and hence not for 2-var quasi-succinct constraints. However, in some situations, *Apriori*<sup>+</sup> may indeed

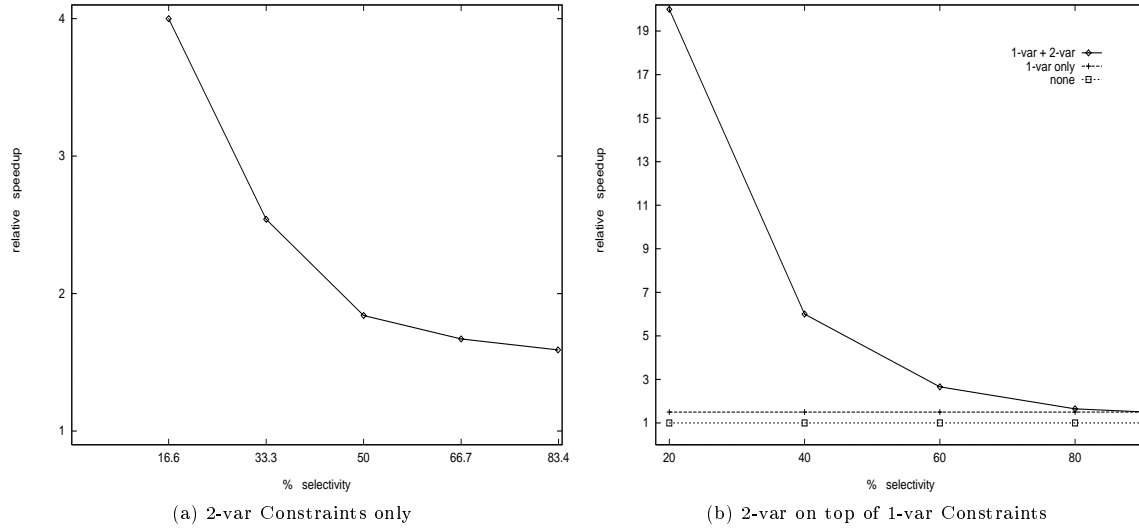


Figure 8: Performance Speedup: Exploiting Quasi-succinctness

be ccc-optimal (as will the strategy given by the optimizer). If  $\mathcal{C}$  consists of only 2-var quasi-succinct constraints where the variables  $S$  and  $T$  effectively point to the same lattice computation, then  $\text{Apriori}^+$  is ccc-optimal. For example, suppose  $\mathcal{C}$  consists of a single constraint  $\{\min(S.A) \leq \min(T.A)\}$  with both  $S, T$  as variables over the domain, say  $\text{Item}$ . The two 1-var constraints given in the first entry of the table in Figure 3 become trivial (e.g.,  $\min(CS.A) \leq \max(L_1^T.A) = \max(L_1^S.A)$ ). Thus, every subset of  $\text{Item}$  is a valid  $S$ - and  $T$ -set! In this case, the strategy used by the optimizer effectively reduces to  $\text{Apriori}^+$ .

As for non-quasi-succinct constraints, the computation strategy given by the optimizer (and trivially,  $\text{Apriori}^+$ ) is not ccc-optimal because the strategy violates both conditions of ccc-optimality. While the two-pronged approach proposed in Section 5 can be effective in cutting down the number of sets counted for support and the number of invocations of constraint checking, it does not guarantee ccc-optimality. Developing ccc-optimal strategies for non-quasi-succinct constraints is an open problem.

## 7 Experimental Evaluation

To evaluate the effectiveness of the various optimizations presented in this paper, we implemented all of them in  $\mathcal{C}$ . We used the program developed at IBM Almaden Research Center [2] to generate the transaction 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. All experiments were run in a time-sharing SPARC-10 environment, and the speedup shown is w.r.t. total CPU + I/O time. For comparisons, we include the results for the “baseline” algorithm  $\text{Apriori}^+$ , which first generates all frequent sets and then checks them for constraint satisfaction. Whenever appropriate, we include the results for the CAP algorithm proposed in [15]. CAP optimizes 1-var constraints by pushing them deeply in an  $\text{Apriori}$ -style bottom-up framework. This algorithm does not optimize the 2-var constraints analyzed in this paper.

### 7.1 Quasi-succinctness: 2-var Constraints Only

In this set of experiments, we consider a single 2-var quasi-succinct constraint  $\max(S.\text{Price}) \leq \min(T.\text{Price})$ . As shown in Figure 3, it can be reduced to the two succinct 1-var constraints:  $\max(CS.\text{Price}) \leq \max(L_1^T.\text{Price})$  and  $\min(CT.\text{Price}) \geq \min(L_1^S.\text{Price})$ . The efficiency gain by using these two 1-var constraints as a replacement of the original 2-var constraint depends on the  $\text{Price}$  ranges of  $S$  and  $T$ . The curve shown in Figure 8(a) corresponds to the case when  $S.\text{Price}$  is in the range  $[400, 1000]$ . (We will comment on the effect on changing this range shortly.) On the other hand,  $T.\text{Price}$  is in the range  $[0, v]$ . The x-axis of the graph in Figure 8(a) shows the results for various values of  $v$ . For easier comparisons among different ranges of  $\text{Price}$ , the x-coordinate is expressed in terms of the percentage overlap between the range of  $S.\text{Price}$  and the range of  $T.\text{Price}$ , i.e.,  $x = 100\% * (v - 400) / (1000 - 400)$ , where  $v \geq 400$ . For instance,  $v$  equal to 500 and 700 correspond to the overlap percentage of 16.6% and 50% respectively. The y-axis shows the speedup of exploiting quasi-succinctness relative to Algorithm  $\text{Apriori}^+$ . The graph in Figure 8(a) shows that the speedup is about 4 times when there is a 16.6% overlap. In general, as there is more overlap, the constraint  $\max(S.\text{Price}) \leq \min(T.\text{Price})$  itself becomes less selective and the speedup is reduced. But even for the large percentage overlap of 83.4%, there is a speedup of over 1.5 times.

The effectiveness of the pruning achieved by exploiting quasi-succinctness is best explained by the following table (for the situation of 16.6% overlap). The columns of the table correspond to the sizes of the frequent sets. The two rows describe the situations for variable  $S$  and  $T$ . Each entry is of the form  $a/b$ , where  $a$  is the number of frequent sets satisfying the corresponding 1-var succinct constraint, and  $b$  is simply the total number of frequent sets of that size. For example, without taking advantage of quasi-succinctness,  $\text{Apriori}^+$  finds 372, 122 and 8 frequent sets of sizes 2, 4 and 6 respectively for variable  $S$ . With quasi-succinctness exploited, our optimized strategy only needs to compute 153,

21 and 1 frequent sets of the corresponding sizes. For variable  $T$ , our optimized strategy stops after 3 levels, whereas Apriori<sup>+</sup> needs to go to 6 levels.

	$L_1$	$L_2$	$L_3$	$L_4$	$L_5$	$L_6$
for $S$	425/425	153/372	54/179	21/122	6/48	1/8
for $T$	402/402	112/414	8/181	0/123	0/48	0/8

The graph shown in Figure 8(a) is based on  $S.Price$  falling in the range [400,1000]. Enlarging (and shrinking respectively) this range makes the constraint less (more) selective and the speedup less (more) prominent. The following table shows the speedup for a percentage overlap of 50%, when the  $S.Price$  falls in the ranges of [300,1000], [400,1000] and [500,1000] respectively.

Range of $S.Price$	Speedup for 50% overlap
[300,1000]	1.52 times
[400,1000]	1.84 times
[500,1000]	2.07 times

## 7.2 Quasi-succinctness: 2-var constraints Together with 1-var constraints

In this set of experiments, we consider the constraints:  $T.Price \leq 600$  &  $S.Price \geq 400$  &  $S.Type = T.Type$ . We compare the relative efficiency of three strategies: (i) the baseline Apriori<sup>+</sup>; (ii) the CAP algorithm which only optimizes the first two 1-var constraints; and (iii) the strategy of exploiting the quasi-succinct 2-var constraint, as well as optimizing the 1-var constraints as in CAP (which is exactly the strategy prescribed by the optimizer). Thus, the difference in performance between the last two strategies is purely based on the way quasi-succinctness is exploited.

The  $x$ -axis of the graph in Figure 8(b) gives the percentage overlap between the  $Types$  of items of  $T$  (i.e., those items satisfying  $T.Price \leq 600$ ) and the  $Types$  of items of  $S$  (i.e., those items satisfying  $S.Price \geq 400$ ). The  $y$ -axis again shows the speedup relative to the Apriori<sup>+</sup> algorithm. This explains the horizontal line at  $y = 1$  in the figure. Because the CAP algorithm only optimizes the 1-var constraints and only checks the 2-var constraint  $S.Type = T.Type$  at the end, the percentage overlap variation has no bearing on the relative performance of CAP. Thus, its performance curve also gives a horizontal line, but this time at  $y = 1.5$ . In other words, optimizing the 1-var constraints alone gives a speedup of 1.5 times. In addition to this optimization, if quasi-succinctness is applied to the 2-var constraint, very significant additional speedup is achieved. For example, for a 40% overlap in  $Type$  values, the total speedup is 6 times, as compared with 1.5 times with only optimizations for 1-var constraints. For a 20% overlap, the total speedup is about 20 times.

The graph shown in Figure 8(b) is based on  $S.Price$  falling in the range [400,1000] and  $T.Price$  in [0,600]. Enlarging these ranges reduces the speedup of both curves relative to Apriori<sup>+</sup>. The reason is that the larger the ranges, the less selective the 1-var constraints are. However, the algorithm CAP, optimizing only 1-var constraints, is more seriously affected by these changes. Consequently, the gap between whether quasi-succinctness is exploited or not is even wider. The following table shows this phenomenon with a 40%

overlap in  $Type$  values. The third and fourth columns of the table give the speedup for optimizing 1-var constraints only, and for optimizing both kinds of constraints respectively. The last column gives the ratio of the fourth column over the third column.

$S.Price$	$T.Price$	Speedup for 1-var only	Speedup for 1- and 2-var	Ratio
[100,1000]	[0,900]	1.2 times	5 times	4.17
[400,1000]	[0,600]	1.5 times	6 times	4.0
[800,1000]	[0,200]	20 times	37.5 times	1.875

## 7.3 Optimizing $sum()$ and $avg()$ Constraints with $J_{max}^k$

Section 5 gives two ways to optimize non-quasi-succinct constraints. The experimental results presented so far already give an idea of the efficiency of the first approach of inducing weaker constraints. Below we focus on the second approach of iterative pruning with  $J_{max}^k$ . In this experiment, we consider  $sum(S.Price) \leq sum(T.Price)$ . Recall from Section 5 that pruning is achieved by finding a series of  $V^2, \dots, V^k$  upper bounding  $sum(S.Price)$ . In order for the series to develop, we pick a low support threshold for  $S$  so that there are frequent sets on the  $S$  side that are of high cardinality, and the effect of the pruning can be appreciated. For the results reported below, the highest cardinality is 14. Furthermore, values of  $S.Price$  and  $T.Price$  are made normally distributed, with different means but the same variance. For the results reported below, the items corresponding to  $S$  have a mean  $Price$  value of 1000 and a variance of 100. The following table shows the speedup with different mean  $T.Price$  values.

Mean of $T.Price$	Speedup with $J_{max}^k$
400	3.14 times
600	1.91 times
800	1.36 times
1000	1.11 times

When the mean  $Price$  value on the  $T$  side is much lower than that on the  $S$  side, the constraint  $sum(S.Price) \leq sum(T.Price)$  is reasonably selective. In this case, the iterative pruning strategy using  $J_{max}^k$  helps bring about a proportionate amount of speedup. For instance, when the mean  $Price$  value on the  $T$  side is 400, the speedup is about 3 times. But when the mean  $Price$  value increases, the constraint itself is less selective. Consequently, the speedup obtained is modest, e.g., only 1.4 times when the mean is 900. Compared with quasi-succinctness, pruning with  $J_{max}^k$  delivers less spectacular results. We attribute this to the relatively non-selective nature of the  $sum()$  and  $avg()$  constraints. *The point is that iterative pruning does deliver a level of performance commensurate with the selectivities of these constraints that are hard to optimize.*

## 8 Conclusions

Towards the eventual goal of supporting ad-hoc mining of various kinds of rules, we proposed in [15] constrained frequent set queries. The main contribution of [15] was in developing pruning optimizations for 1-var constraints. In this paper, we consider 2-var constraints and develop pruning

optimizations for them. We establish a negative result that few 2-var constraints are anti-monotone, thus underscoring the challenge posed by 2-var constraints w.r.t. pruning optimization. We introduce the notion of quasi-succinctness and completely characterize the class of all such constraints among those allowed in the CFQ language. Quasi-succinct constraints can be reduced to two succinct 1-var constraints which are sound and tight w.r.t. pruning away candidate  $S$ -sets and  $T$ -sets. For constraints that are not quasi-succinct, we develop a two-pronged approach consisting of: (i) inducing weaker quasi-succinct constraints and exploiting them, and (ii) adopting an iterative pruning strategy. Finally, we propose a query optimizer for CFQs and show that for the large class of 1-var succinct and 2-var quasi-succinct constraints, the strategy generated by the optimizer achieves the very desirable goal of ccc-optimality. This notion captures the idea that the effort spent by the strategy in invoking support counting and constraint checking is minimized. We establish the effectiveness of the optimizations developed in the paper with experiments, which show significant speedup, compared with Apriori<sup>+</sup> on the one hand, and compared with CAP (which optimizes only 1-var constraints) on the other.

Many questions remain open, but we mention three here. (1) As mentioned in Section 5, the strategies developed for non-quasi-succinct constraints are not ccc-optimal. Developing such a strategy is an open problem. (2) The “cost model” corresponding to ccc-optimality represents only a first attempt to explore optimality issues. Developing more detailed cost models for CFQs, as well as optimizers incorporating such models, is an interesting problem. (3) Expanding the constraint language to incorporate more powerful, yet useful, constraint classes is another important problem.

## References

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD*, pp 207–216.
- [2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 VLDB*, pp 487–499.
- [3] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. 1998 ACM-SIGMOD*, pp 85–93.
- [4] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. In *Proc. 1997 ACM-SIGMOD*, pp 265–276.
- [5] S. Chaudhuri. Data mining and database systems: Where is the intersection? *Bulletin of the Technical Committee on Data Engineering*, 21:4–8, March 1998.
- [6] D.W. Cheung, J. Han, V. Ng, and C.Y. Wong. Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proc. 1996 ICDE*, 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. In *Proc. 1996 ACM-SIGMOD*, pp 13–23.
- [8] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 VLDB*, pp 420–431.
- [9] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of ACM*, 39:58–64, 1996.
- [10] M. Kamber, J. Han, and J. Y. Chiang. Metarule-guided mining of multi-dimensional association rules using data cubes. In *Proc. 3rd KDD’97*, pp 207–210.
- [11] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A.I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proc. 3rd Int. Conf. Information and Knowledge Management*, pp 401–408, 1994.
- [12] F. Korn, A. Labrinidis, Y. Kotidis, and C. Faloutsos. Ratio rules: A new paradigm for fast, quantifiable data mining. In *Proc. 1998 VLDB*, pp 582–593.
- [13] L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries: 2-var constraints. Technical Report, Department of Computer Science, University of British Columbia, 1998.
- [14] R.J. Miller and Y. Yang. Association rules over interval data. In *Proc. 1997 ACM-SIGMOD*, pp 452–461.
- [15] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 ACM-SIGMOD*, pp. 13–24.
- [16] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In *Proc. 1995 ACM-SIGMOD*, pp 175–186.
- [17] S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. In *Proc. 1998 VLDB*, pp 368–379.
- [18] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems: Alternatives and implications. In *Proc. 1998 ACM-SIGMOD*, pp 343–354.
- [19] A. Silberschatz and S. Zdonik. Database systems – breaking out of the box. *SIGMOD Record*, 26, pp 36–50, 1997.
- [20] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. In *Proc. 1998 VLDB*, pp 594–605.
- [21] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. 1995 VLDB*, pp 407–419.
- [22] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. 1996 ACM-SIGMOD*, pp 1–12.
- [23] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *Proc. KDD’97*, pp 67–73.
- [24] H. Toivonen. Sampling large databases for association rules. In *Proc. 1996 VLDB*, 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. In *Proc. 1998 ACM-SIGMOD*, pp 1–12.