On Dual Mining: From Patterns to Circumstances, and Back

Gösta Grahne*, Laks V.S. Lakshmanan*†, Xiaohong Wang*, and Ming Hao Xie*
* Concordia University † IIT - Bombay
{grahne, laks, wang_x, mhxie}@cs.concordia.ca

Abstract

Previous work on frequent itemset mining has focused on finding all itemsets that are frequent in a specified part of a database. In this paper, we motivate the dual question of finding under what circumstances *a given itemset satisfies a pattern of interest (e.g., frequency) in a database. Circumstances form a lattice that generalizes the instance lattice associated with datacube. Exploiting this, we adapt known cube algorithms and propose our own,* minCirc*, for mining the strongest (e.g., minimal) circumstances under which an itemset satisfies a pattern. Our experiments show minCirc is competitive with the adapted algorithms. We motivate mining queries involving migration between itemset and circumstance lattices and propose the notion of* Armstrong Basis *as a structure that provides efficient support for such migration queries, as well as a simple algorithm for computing it.*

1 Introduction

Extensive work has been done on efficient mining of interesting patterns like associations, correlations, etc. and their variants, from large databases (e.g., see [AS94, MTV94, BMS97, LSW97, AS95, MTV97, DL99, HPY00, PHM00, Bay98, AAP00, ZaHs99]). At the core of extracting such patterns is the determination of frequent itemsets, to which many of these studies are devoted. All of these studies are concerned with answering the question "which patterns hold in the database." We argue that many a time, it is at least as interesting and useful to ask "*when* a given pattern holds in a database." Here the word *when* can be interpreted to mean, but need not be restricted to, temporal conditions, conditions involving locations, demographics, etc., and more generally, the "underlying circumstances" under which the pattern of interest holds in the database.

Why should we care about this question? Here are a few motivating example queries.

Q_1: Under what circumstances (say, on time of purchase) is the itemset {beer, diaper} bought frequently?

Q_2: Under what circumstances (e. g. time, location) are interstate calls totalling over $50 made by a customer?

Q_3: Under what circumstances (e. g. season, weather) are ice cream sales high? Which items(ets) have high sales under those circumstances?

Q_4: Which itemsets are seasonal favourites, i.e. they are bought frequently in fall but not in summer or vice versa?

Q_5: For each itemset bought frequently in fall, find further circumstances under which they are bought frequently.

Just as associations, correlations, and other mined patterns are useful in sales planning etc., knowing the circumstances under which one or more patterns hold can be useful in "tuning" the plans around these circumstances. E.g., if certain patterns hold for NJ locations and not for NY locations, then a manager might want to incorporate this in his planning. The examples above illustrate that one might be interested in a specific pattern (e.g., a specific frequent itemset {beer, diaper} in Q_1), or in a set of patterns generated from another query (e.g., the set of frequent itemsets bought in fall in Q_4), and ask for further circumstances under which they hold.

A second observation is that for certain patterns, if they hold under a given circumstance c, then they necessarily hold in a weaker circumstance c', where weaker means c implies c'. As an example, consider frequent itemset as a pattern. Let c be the circumstance $\texttt{city}$ = 'westfield' & $\texttt{state}$ = 'nj' & $\texttt{month}$ = 'october' and c' be $\texttt{state}$ = 'nj'. Clearly, c' is weaker than c in the sense that every transaction that satisfies c necessarily satisfies c'. Equivalently, c is stronger than c'. Let S be any itemset. Suppose S is frequent under c. Then it must be frequent under c'. [1] In the terminology of Ng et al. [NLHP98, LNHP99], we call such patterns *monotone*. For these patterns, there is an intrinsic interest in determining the "strongest" circumstances under which they hold. E.g., suppose we know the set of strongest circumstances $\mathcal{C}$ under which an itemset S is frequent. Intuitively, this covers the space of all circumstances under which S is frequent, since we know S is frequent exactly under those circumstances that are weaker than some circumstance in $\mathcal{C}$. Another example of a pattern that

[1] Throughout this paper, we use the absolute support count for frequency. This count can come from a specified fraction, say 0.1%, of the *total* database size, for example.

has this property occurs in (Q2): if interstate calls totaling over $50 were made under a given circumstance (e.g., over a week), then under a weaker circumstance (e.g., over a month containing that week), the total cannot be less.

Some patterns tend to be *anti-monotone* in that whenever they hold in a circumstance, they necessarily hold in a stronger circumstance. A natural example is the pattern "interstate calls made total under $25." Clearly, if this holds for a given circumstance, it must hold for any stronger circumstance[2]. For such patterns, following the motivation similar to the previous paragrahph, there is an intrinsic interest in asking for the weakest circumstances under which they hold.

The rest of this paper is organized as follows. In Section 2 we formalize the notions item, transaction, and circumstance. We show in Section 3, that for many applications, the space of circumstances forms a lattice, which we call the *circumstance lattice*. This lattice generalizes the cube lattice, often employed by algorithms for computing the data cube operator proposed by Gray et al. [G*96]. (See also [BR99], e.g.) In Section 4 we develop an algorithm, called *minCirc*, for determining the strongest (resp., the weakest) circumstances under which patterns satisfying either the monotone or the anti-monotone property hold. We also establish a close relationship between the problem of finding all circumstances under which a given pattern holds and data cube computation, and we make obvious adaptations to two representative popular algorithms for cube, including BUC ([BR99]) and chunked array cube ([ZDN97]). In Section 5 we experimentally evaluate the relative performance of all three algorithms. Our results show algorithm *minCirc* is competitive in that there are conditions under which each algorithm (*minCirc* included) is the algorithm of choice.

Some of the motivating queries above go beyond merely finding circumstances under which a pattern holds. To capture this class of queries, in Section 6, we propose a notion called *Armstrong Basis* and show that a variety of useful queries can be answered from this basis. Section 7 discusses related work, and summarizes the paper.

2 Problem Definition

Traditional framework of association mining regards each attribute-value pair as an "item" and attempts to discover associations between sets of items using measures such as support and confidence. Given a set of transactions, not all attributes are alike. Some attributes may have a unique value per transaction while others may have multiple values in a given transaction. As an example, items bought from a supermarket naturally corresponds to an attribute that has a set of values per transaction. On the other hand, the timestamp associated with a transaction typically has a unique value per transaction. While treating all attributes alike is certainly possible, we contend that it misses out on the opportunities of exploring and exploiting the different structures and properties associated with dissimilar attributes. E.g., for an attribute such as time, different values are mutually exclusive in that a transaction can only have one value for it. By distinguishing among such attributes, we can formulate queries like "what are the intervals during which there is a significant volume of sales on dessert items". As another example, we can ask "which are the locations where there is a strong association among a set of items being purchased". We contend that mining queries of this form can be best facilitated by developing a framework where attributes that are inherently like items (we call them "item attributes" below), i.e. have a set of values per transaction, are distinguished from those that are more like time (we call them "circumstance attributes" below), which have a unique value per transaction. The potential "item attributes" need further to be distinguished from "descriptive attributes," such as cost or quantity. These considerations motivate the following definitions.

Consider a table over a set of attributes. We wish to regard tuples, or possibly sets of tuples, in this table as transactions. To this end, we can designate any set of attributes in the table as a transaction id, as long as this attribute set uniquely determines the transaction. E.g., the set $\texttt{customerID}, \texttt{timeID}$ might constitute the transaction id for the transactions for a table containing a supermarket sales data. As another example, $\texttt{locationID}, \texttt{prodID}, \texttt{timeID}$ might serve as a transaction id for tuples in a fact table of a data warehouse containing sales data. In general, given a table r over a set of attributes R, we assume that the user specifies a subset of attributes, say $K \subset R$, as its *transaction id*. This induces a partition on the table with tuples sharing the same K-value being in the same cell of the partition. [3] We call an attribute $A \in R \setminus K$ *circumstance attribute* provided the functional dependency $K \rightarrow A$ holds for r. For other attributes $A \in R \setminus K$ it follows from the axioms of functional and multivalued dependencies that the multivalued dependency $K \rightarrow\!\!\rightarrow Y$, where $A \in Y$, will hold in r. If in addition the functional dependency $KA \rightarrow Y \setminus A$ holds we call A an *item attribute*, otherwise we call A a *descriptive attribute*.

As an example, consider a table $\texttt{sales}(\texttt{tid}, \texttt{location}, \texttt{time}, \texttt{product}, \texttt{quantity})$ with $\texttt{tid}$ acting as the transaction id, in the sense that each $\texttt{tid}$-value uniquely determines a set of tuples, in which a set of products are sold at a given time and location. We also have $\{\texttt{tid}\} \rightarrow\!\!\rightarrow \{\texttt{product}, \texttt{quantity}\}$, and $\{\texttt{tid}, \texttt{product}\} \rightarrow \{\texttt{quantity}\}$. Thus $\texttt{product}$ is an item attribute and $\texttt{quantity}$ is a descriptive attribute, while $\texttt{location}$ and $\texttt{time}$ are circumstance attributes. [4]

An *itemset* is then a set of attribute-value pairs, where

[2] The strongest circumstace is *false*, which verifies any pattern vacuously.

[3] Each cell might be a singleton set.

[4] In this paper we do not focus on descriptive attributes. In the constrained data mining framework of [LNHP99, GLW00], constraints would typically be formulated over circumstance and descriptive attributes.

the attribute is an item attribute and the value comes from its domain. By a *circumstance*, we mean a conjunction of predicates of the form $attr\theta value$, where *attr* is a circumstance attribute, and *value* comes from its domain. Here, typically, we want to set θ to be equality. When dealing with totally ordered circumstance attributes (e.g., time, distance, etc.), we want to allow θ to be $\leq$, since on such domains, intervals over which patterns hold are of interest. A transaction *satisfies* a circumstance ψ, provided it satisfies the predicate ψ.

Given a support threshold s, [5] we say that an itemset S is *frequent* in a transaction database provided at least s transactions contain S. More generally, given an itemset S and a circumstance ψ, we say that S is frequent under the circumstance ψ provided s transactions among those satisfying the circumstance ψ contain S. [6] The traditional notions of support and confidence in association mining can be relativized to circumstances: the *support* of S under ψ is the number of transactions satisfying ψ that contain S; for itemsets S and T, the *confidence* of a rule $S \Rightarrow T$ under ψ is the proportion, among the transactions satisfying ψ, the number of transactions containing $S \cup T$ over those that contain S.

In data mining, there are various notions of interesting patterns, such as frequent itemsets, associations, correlations, etc. In the bulk of this paper, we shall focus on frequent itemsets and address the following question: *Given a specific itemset S, a support threshold s, find all circumstances under which S is frequent.*

The reason we pick frequency as a representative pattern for detailed study is that it is the simplest among those discussed and yet epitomizes the issues that arise in studying other patterns. Also, as discussed previously, frequent itemsets are often used as a basis for finding other patterns like associations, correlations, etc.

Later, we will address how techniques developed for solving the above problem can be adapted for handling associations.

3 Structure of Circumstances

In general, the space of circumstances forms a lattice, called the *circumstance lattice*. To see this, recall that circumstances are conditions comparing attributes with values. Define a partial order $\leq$ on the set of all circumstances Ψ, by setting $\psi \leq \phi$ exactly when ψ logically implies ϕ. It is well-known that a collection of sentences ordered this way forms a lattice. In particular, *true* is the top element and *false* is the bottom element. Figure 1 shows an example circumstance lattice for two circumstance attributes A, B each with two values in its domain. For simplicity, the figure only considers circumstances involving equality conditions.

Circumstance attributes sometimes come with hierarchies. E.g., locations may have the hierarchy `storeId-city-state`. It is a straightforward matter to see that the

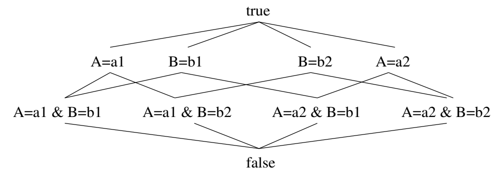

Figure 1. An Example Circumstance Lattice

notion of lattice extends to circumstance attributes with hierarchies, as well as to circumstances involving inequalities. *In the rest of the paper, unless otherwise specified, we only consider circumstances involving equality conditions and no attribute hierarchies.* More general conditions are left for future work.

Let $(\mathcal{I}, \subseteq)$ denote the itemset lattice for a given application and $(\mathcal{C}, \leq)$ the circumstance lattice w.r.t. the circumstances pertaining to the application. We can construct a new structure $\mathcal{C} \times \mathcal{I}$ by taking the direct product of the two lattices. On this structure, we can define two alternative orders: the first order $\leq^{\mathrm{up}}_{\mathrm{up}}$ is defined as $(\psi, S) \leq^{\mathrm{up}}_{\mathrm{up}} (\psi', S')$ iff $\psi \leq \psi'$ and $S \subseteq S'$; the second order $\leq^{\mathrm{up}}_{\mathrm{down}}$ is defined as $(\psi, S) \leq^{\mathrm{up}}_{\mathrm{down}} (\psi', S')$ iff $\psi \leq \psi'$ and $S' \subseteq S$. A structure $(\mathcal{L}, \leq_1, \leq_2)$ with two partial orders is called a *bilattice* [Fit91], provided each of $(\mathcal{L}, \leq_1)$ and $(\mathcal{L}, \leq_2)$ is a lattice, and a certain condition relating the two orders hold. It can be shown that $(\mathcal{C} \times \mathcal{I}, \leq^{\mathrm{up}}_{\mathrm{up}}, \leq^{\mathrm{up}}_{\mathrm{down}})$ is actually a bilattice, which we call the *circumstance-itemset bilattice*.

The bilattice formulation gives a semantic basis for known mining problems and inspires newer ones too. For example, the classical problem of mining frequent itemsets from a given transaction database reduces to finding all itemsets S such that S is frequent under *true*. More generally, if we wanted to find all frequent sets in a selection view (not involving aggregates) defined by conditions over the transaction database, this would amount to finding the set $\{S \mid S \text{ is an itemset frequent under } \psi\}$, where ψ is the conjunction of conditions defining the view. Let $\mathcal{F}$ denote the subset of the bilattice containing precisely those pairs (ψ, S) such that S is frequent under ψ. Then frequent set mining given ψ corresponds to finding $\{S \mid (\psi, S) \in \mathcal{F}\}$. Circumstance mining for a given itemset S corresponds to finding the set $\{\psi \mid (\psi, S) \in \mathcal{F}\}$. The bilattice structure is instrumental in shaping the concept of an Armstrong Basis, the main object of Section 6.

In the rest of this section, we consider general patterns involving itemsets and classify them according to the properties they satisfy. Since we want to do this in relation to circumstances, we need some notation first. We define a *pattern* over the bilattice $(\mathcal{C} \times \mathcal{I})$ as any predicate over it, i.e. any subset of the elements of the bilattice. E.g., frequency corresponds to the collection of elements $(\psi, S) \in (\mathcal{C} \times \mathcal{I})$, such that S is frequent under ψ; correlation corresponds to the collection of elements (ψ, S) such that S is a correlated set under ψ. As another example, consider the statement $minpurchase(S)$ that says the total dollar value

[5] Recall, we use an absolute number.

[6] Thus, if fewer than s transactions satisfy ψ, S is infrequent under ψ.

of the purchase on items in S must exceed a threshold, or $purchasecap(S)$ that says the total dollar value of the purchase on items in S must be below a threshold. Each of these defines a subset of the bilattice. We use the notation $p_{\mathcal{D}}(\psi, S)$, or simply $p(\psi, S)$, to say pattern p holds for the element (ψ, S) in database $\mathcal{D}$. E.g., when p is *purchasecap*, this means over those transactions satisfying circumstance ψ, the total dollar value of purchase of items in S does not exceed the given threshold.

We say that a pattern p is *c-monotone*, provided whenever $p(\psi, S)$ holds, $p(\phi, S)$ holds as well, for every $\psi \leq \phi$. It is *c-anti-monotone*, provided whenever $p(\psi, S)$ holds, $p(\phi, S)$ holds as well, for every $\phi \leq \psi$. The notions *i-monotone* and *i-anti-monotone* are defined analogously and correspond exactly to what was referred to as *monotone* and *anti-monotone* in the frequent set mining literature [NLHP98]. In general, a pattern may be monotone or anti-monotone w.r.t. either lattice. For example, frequency is c-monotone and i-anti-monotone. The pattern *purchasecap* is c-anti-monotone and i-anti-monotone, while *minpurchase* is i-monotone and c-monotone. Being correlated as defined in [BMS97] is i-monotone but neither c-monotone nor c-anti-monotone.

Many of these patterns satisfy interesting identities. The following Proposition is proved using the axioms of bilattices.

Proposition 1 *Let p be any pattern. Then we have the following.*

1. *if p is i-anti-monotone, then $\{\psi \mid p(\psi, S)\} \cap \{\psi \mid p(\psi, T)\} \subset \{\psi \mid p(\psi, S \cap T)\}$; a similar identity holds w.r.t. union.*
2. *if p is c-monotone, then $\{S \mid p(\psi \wedge \phi, S)\} \subset \{S \mid p(\psi, S)\} \cap \{S \mid p(\phi, S)\}$; a similar identity holds w.r.t. disjunction.*
3. *similar identities holds for i-monotone and c-anti-monotone patterns.*

As an example, frequency satisfies the two identities in the proposition. On the other hand, *purchasecap* satisfies the first identity as well as the identity: $\{S \mid purchasecap(\psi, S)\} \cap \{S \mid purchasecap(\phi, S)\} \subset \{S \mid purchasecap(\psi \wedge \phi, S)\}$.

The identities above illustrate what kind of pruning opportunities exist when processing complex queries. For instance, suppose we want to find itemsets that are frequently bought in northeast and in fall, and that we already have itemsets corresponding frequent purchases in northeast and frequent purchases in fall. Then one way of processing this query is to verify each itemset in the intersection for frequency in the circumstance "northeast and fall." A strategy like this makes sense when we want to process queries corresponding to circumstances at multiple granularities.

4 Algorithms for Circumstance Mining

In this section, we address the problem of finding all circumstances in which a given pattern holds for a given itemset, when the pattern satisfies some monotonicity properties. For the sake of concreteness, we pick frequency as a prototypical example. Since it is a c-monotone pattern, knowing an itemset is frequent under a circumstance tells us it is frequent under all circumstances implied by it. Thus, we really wish to find the *minimal* circumstances under which a given itemset is frequent, in the sense that the itemset is not frequent in any stronger circumstance. For c-anti-monotone patterns such as *purchasecap*, the set of *maximal* circumstances in which the pattern holds w.r.t. a given itemset summarizes the whole space of circumstances where this pattern holds for that itemset. In the rest of this section, we focus on frequency, a c-monotone pattern.

We note that the circumstance lattice bears a strong similarity to the lattice used for data cube computation. Indeed, when we limit ourselves to circumstances involving only equality, the circumstance lattice is identical to the lattice corresponding to the (instances of) various group-bys. This suggests any of the algorithms developed for cube should be useful for mining circumstances. On the other hand, given the c-monotone nature of frequency, a traditional Apriori-style level-wise pruning strategy should be relevant as well, except we use the circumstance lattice in place of the itemset lattice. Rather than trying to be exhaustive, in this section, we will discuss simple adaptations to *some* existing cube algorithms, as well as propose a new algorithm, for finding minimal circumstances under which a given itemset is frequent.

The representative cube algorithms we pick are the BUC (bottom-up cube, which is actually top-down in our lattice formulation) algorithm proposed by Beyer and Ramakrishnan [BR99] and the chunk array-based algorithm (which is bottom-up in our formulation) due to Zhao et al. [ZDN97]. The main efficiency of chunk array-based algorithm comes from using a multi-dimensional address space for storing the base table and the cube itself. This affords great compression. It uses the minimum weight spanning tree data structure to optimize cube computation. In addition, it also uses techniques for handling sparse data. Figure 2 shows the adapted chunk array algorithm for finding minimal circumstances in which a given itemset S is frequent. The basic idea is we perform a search in the circumstance lattice from the strongest circumstances to the weaker ones. As soon as S is found to be frequent in a circumstance, we know it must be minimal. Clearly, all minimal circumstances can be found in this way. The correctness follows from these observations.

The BUC algorithm was also designed to incorporate constraints on groups such as $count(*) > n$, which says the group must have at least n elements. However, the express goal of BUC was to find *all* groups satisfying given constraints (as opposed to minimal ones). Figure 3 shows our adaptation to the BUC algorithm for finding minimal circumstances. A minor modification consists in computing the support of S in every group (i.e. circumstance). Just like original BUC, whenever S is found to be infrequent in

Algorithm Modified Chunk Array;
Input: an itemset S, a support threshold s, and a transaction database $\mathcal{D}$;
Output: the set of minimal circumstances in which S is frequent;
Run chunk array algorithm, with the following modifications:

1. no aggregate computation (other than verifying the support constraint) is needed;
2. when processing a circumstance ψ, compute the support of S in ψ; if it exceeds s, add ψ to the output; circumstances that are implied by ψ are pruned;

Figure 2. Adaptation to Chunk Array Algorithm for finding minimal circumstances.

Algorithm Modified BUC;
Input: an itemset S, a support threshold s, and a transaction database $\mathcal{D}$;
Output: the set of minimal circumstances in which S is frequent;

1. run BUC algorithm, with the following modifications:
   (a) no aggregate computation (other than verifying the support constraint) is needed;
   (b) when processing a circumstance ψ, compute the support of S in ψ; if it is below s, circumstances that are imply ψ, including ψ, are pruned; otherwise, ψ is retained;
2. do post-processing to eliminate non-minimal circumstances;

Figure 3. Adaptation to BUC Algorithm for finding minimal circumstances.

a circumstance, we prune away all stronger ones. A final modification consists in post-processing: BUC (modified as above) finds all circumstances in which S is frequent; we need to efficiently detect and eliminate non-minimal ones. Correctness of modified BUC is straightforward.

Post-processing: Circumstances are processed in increasing order of their length (e.g., $A = a_1 \wedge B = b_1$ is of length 2) in BUC. In addition, within each length, we can make sure they are processed in lexicographical order (by assuming some arbitrary, but fixed order on circumstance attributes and their domains). Let n be the number of circumstance attributes. Then all circumstances of length n found by the modified BUC algorithm are minimal, by definition. For each circumstance of length $k < n$, we check if it is implied by any circumstance of length $k+1$. The implication test is syntactic and checking length $(k+1)$-circumstances for implication can be done efficiently using binary search.

The last algorithm, called *minCirc*, that we propose in this section is in some sense inspired by Apriori. The basic idea is since frequency is c-monotone, we can scan the transactions repeatedly and compute the support of S corresponding to all circumstances of a given length in each iteration, starting from length 1. When S is infrequent in a circumstance, we prune all circumstances of which it is a "prefix". E.g., when S is infrequent in $A = a_1 \wedge B = b_1$, we ignore $A = a_1 \wedge B = b_1 \wedge C = c_1$, etc. A literal implementation of this idea, however, can lead to a poor performance. The reason is when we have several circumstance attributes, each with a domain of reasonable cardinality, the effort required for candidate generation (even with this pruning) is appreciable. Our algorithm instead imposes an order on the dimensions and exploits this order in quickly pruning away

Algorithm minCirc;
Input: an itemset S, a support threshold s, and a transaction database $\mathcal{D}$;
Output: the set of minimal circumstances in which S is frequent;

1. choose some circ attribute order, say $D_1, \ldots, D_n$;
2. scan $\mathcal{D}$ once and obtain counts of S in all atomic circumstances, i.e. in all D_1-circumstances, ..., D_n-circumstances;
3. let F_1 be the set of all atomic circumstances in which S is frequent;
4. hash the circumstances and write the counts;
5. create a linked list of candidates as follows:
   (a) for $(1 <= i <= n; i++)$ {
      let $x_i \in F_1$ be any D_i-value;
      create a linked list with head x_i;
      for $(i < j <= n; j++)$ {
         let $x_j \in F_1$ be any D_j-value;
         append a node containing x_j to the linked list with head x_i; } }
   (b) $k = 2$;
   (c) while the current set of linked lists is non-empty {
      scan $\mathcal{D}$ and obtain counts of all k-circumstances appearing in the linked lists;
      purge each node corresponding to a circumstance in which S is infrequent;
      construct linked lists for $(k+1)$-circumstances (candidates) as follows:
      for each linked list L with head H of size k {
         if $(length(L) >= 2)$ {
            for each node (containing a circ) x in L {
               temporarily create a new list with head Hx, which contains all nodes of L after node x, that correspond to later circumstance attributes;
               //since $length(L) >= 2$, there will be at
               //least one such node;
               delete from this list those nodes y such that some k-subset of Hxy does not appear in any linked list with head size k;
               //this test can be performed quickly by sorting;
               } } } }
6. eliminate non-minimal circumstances via post-processing;

Figure 4. A linked list based mining algorithm for finding minimal circumstances.

many candidate circumstances early. As an example, if S is found to be frequent in $A = a_1 \wedge D = d_1$, where D is the last circumstance attribute, there is no need to check "suffixes" of this circumstance. Figure 4 shows the algorithm. It makes use of the following notions.

Suppose $D_1, ..., D_n$ is a chosen order on the circumstance attributes and let c and d be values of any circumstance attributes. Then $c \prec d$ iff c is a D_i-value and d is a D_j-value, for some $i < j$; c and d are said to be incomparable otherwise. We sometimes find it convenient to write circumstances such as $A = a_1 \wedge B = b_1 \wedge C = c_1$, simply as $a_1b_1c_1$, relying on the fixed chosen order of circumstance attributes. We use linked lists with heads and nodes (head is not a node). For a linked list, its length represents the number of nodes it has. Each list represents a set of candidate circumstances of a given length. E.g., a list with head a_1b_1 and nodes $c_1, \ldots, c_5, d_2, d_6, d_7$ (where $c_i \in dom(C)$ and $d_j \in dom(D)$) represents the set of circumstances $a_1b_1c_1, a_1b_1c_5, \ldots, a_1b_1d_7$. We say a circumstance $x_1 \cdots x_k$ appears in a linked list provided its head contains $x_1 \cdots x_{k-1}$ and one of its nodes contains the circ x_k. In the algorithm, we refer to a circumstance in which S is frequent as a frequent circumstance.

The last step of the algorithm invokes post-processing, discussed earlier. In particular, since the circumstances are processed in increasing order of length, the same post-processing as used for BUC can be used. The correctness of the algorithm follows from the following facts: (i) a circumstance is never explicitly pruned unless S is infrequent in it; (ii) circumstances that are suffixes of those ending in the last circumstance attribute value are never considered for counting, e.g., if we have the circumstance attribute order $ABCD$, then we do not consider suffixes of $b_1 d_1$, for instance; every suffix of such circumstances are covered by some circumstances that we do consider; e.g., the suffix $b_1 d_1 c_1 \equiv b_1 c_1 d_1$, and one of its (not necessarily proper) prefixes, e.g., b_1, is always considered by the algorithm.

Relevance to Cube: A significant feature of the chunk array and BUC algorithms (with modifications) is that they can be used to compute not only (minimal) circumstances where a pattern holds for an itemset, but also any required aggregate measure value at each such circumstance. What can we say about the algorithm minCirc, inspired by Apriori-style intuition? It turns out by associating a field for computing the aggregate measure with each node, we can indeed compute the aggregate incrementally. This argument holds for all distributive and algebraic aggregate functions, for which most known fast cube algorithms have been developed. This establishes an interesting tight connection between frequent set mining and cube computation. Our framework is applicable not just to frequent sets but to more general itemset patterns with monotone or anti-monotone properties w.r.t. itemsets or circumstances.

5 Experimental Results

We conducted a series of experiments to evaluate the effectiveness of the algorithms presented for mining minimal circumstances in which an itemset is frequent. More specifically, we compared the three algorithms to find out how they perform under different conditions. For the given three algorithms, BUC is a top-down algorithm[7] and uses a "depth first" principle; minCirc is also a top down algorithm but uses level-wise principle; Chunk Array by nature is a bottom-up algorithm and it also uses the level wise principle. So these three algorithms represent three different approaches to mining minimal circumstance sets from different directions of the circumstance lattice, and using different search strategies. We want to compare the performance of the algorithms not only by total cpu time, but also by the number of candidate sets the algorithms generated and tested.

A Pentium II 200 processor with 64 KB memory is used for all the experiments. All the algorithms are implemented in C++ running on a Windows NT platform. For test data we generated synthetic sets of transactions for various cases. We used a Zipf distribution to control the skewness of the data, as is commonly used in experiments of this kind, see e. g. [BR99]. To be fair to all the algorithms, we tested both the cases where the transaction database fits in main memory and where it is disk-resident. We used 1 to 7 circumstance attributes, and each attribute had a domain of cardinality 100. The number of transactions varied from 10,000 to 500,000, the support threshold from 5 to 1000, and skewness was varied from 0, which is uniform distribution, to 3. In each of the experiments, the number of candidate sets generated, the peak memory allocation, the total time, the I/O time, and the number of passes over the transaction database was recorded for analysis. For lack of space, we only show the results from a representative subset of our experiments.

In the first set of experiments we used transaction databases which fit in main memory. All other parameters were varied as described above. We measured the total cpu time as a function of the number of transactions. The results were fairly uniform, and a typical representative of this set is shown in Figure A. There we can see that with the increase of the number of transactions, there is a linear increase in the total time for all three algorithms. But the slope for minCirc is steeper than the slope for BUC and Chunk Array. This is due to the fact that BUC and Chunk Array make only one pass over the transaction database, whereas minCirc makes up to as many passes as the number of circumstance attributes.

The second set of experiments (Figures B – E) conducted is for disk-resident transaction databases. When the data is uniformly distributed (skewness = 0), minCirc gives the best performance among the three algorithms. Figure B displays the results for support threshold at 100, and number of circumstance attributes at 5. When the number of transactions increases, the total time increases rapidly for BUC. When the number of transaction is equal to 500,000, the total time for BUC is 50 times larger than for minCirc. Even for Chunk Array the total time is 5 times larger than minCirc. For minCirc, the upper bound on the number of passes over the transaction database equals the number of circumstance attributes. On the other hand, for BUC the only upper bound is the size of the circumstance lattice. Furthermore, as the data is uniformly distributed, Chunk Array has to do many more computations for each of the atomic circumstance sets, so minCirc also performs better than Chunk Array under these conditions.

However, with the increase of the skewness, as the data is not uniformly distributed, the performance of Chunk Array will improve quite drastically since some of the atomic circumstance sets will not appear in the transaction database. Figure C gives the results for the same condition as Figure B, except that the skewness is equal to 1. Similar performance can be observed for BUC. Under these conditions, Chunk

[7]See remark in the beginning of Section 4

Array is the algorithm of choice. [8]

Another factor that affects the performance of these three algorithms, especially Chunk Array, is the number of circumstance attributes. In our case, when the number is 5, and the domain of each circumstance attribute has cardinality 100, there is potentially 10^{10} atomic circumstance sets appearing in the transaction database. When we increase the number of circumstance attributes to 6, there might be up to 10^{12} atomic circumstance sets. This is a large increase for Chunk Array, but for minCirc this just implies at most one more pass over the transaction database. For these conditions minCirc becomes the best choice. Figure D shows the results for 6 circumstance attributes. When the number of transactions is smaller than 300,000, the total time for BUC and Chunk Array is similar, and both about twice the total time of minCirc. When the number of transactions increases to 500,000, the total time for minCirc is 5 times smaller than the other two algorithms.

Another interesting experiment to do is to observe the effect of varying number of circumstance attributes (Figure E). When the number is smaller than 5, Chunk Array has the best performance and the slope of the curve is very gradual. However, once the number of circumstance attributes is greater than 6, there is a dramatic decrease in the performance of Chunk Array and it becomes the worst among the three algorithms. With the increase in the number of circumstance attributes, the total time for both BUC and minCirc increase much more slowly when compared to Chunk Array.

In the third and last set of experiments (Figures F – H) we tested the effect of the skewness of the data and the possible reasons for an increase in the number of candidate sets generated. When the data is uniformly distributed and is disk-resident, minCirc has the best performance and BUC gives the worst performance. When skewness is greater than 1, the curve for all three algorithms become flat. Chunk Array shows the best performance while BUC still is the worst one. In another experiment, not shown here, we tested main memory resident transaction databases, while keeping the other parameters as in Figure F, we observed BUC almost unaffected by the skewness of the data, and gives the best performance among the three algorithms.

In the final experiments of this set, we tested the number of candidate sets generated. This measure is not affected by the residency of the transaction database. The skewness of the data does affect the performance, but not the relative differences between the three algorithms. We show two experiments, namely Figure G and H for the number of candidate sets generated vs. the number of transactions. Skewness is at 1, and the number of circumstance attributes is 5. Figure G shows the result for the threshold at 1000, and Figure H shows the result for the threshold at 5. When the threshold is high, the pruning strategies for BUC and minCirc are very effective, so at 500,000 transactions the number of candidate sets generated by minCirc is over 500 times smaller than that of Chunk Array, and BUC generates 80 times fewer candidate sets than Chunk Array. However, when the threshold is small, the pruning strategy for BUC loses its efficiency, and the number of candidate sets becomes the largest. minCirc also loses some of its pruning efficiency, but it still comparable to Chunk Array.

The prescriptions emerging from our experiments are described in the decision tree of Figure 5.

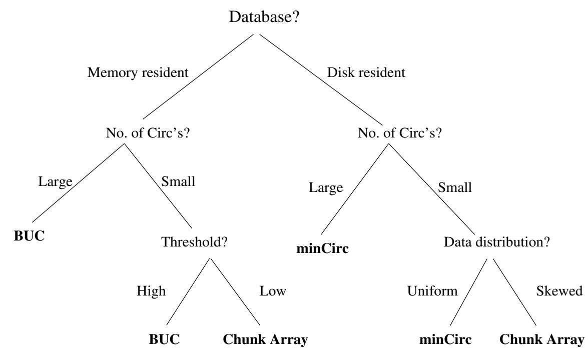

Figure 5. Decision tree for choosing a mining algorithm.

6 Beyond Circumstance Mining

So far in the paper, we have been concerned with finding circumstances where a pattern holds for a given itemset. In general, mining is exploratory in nature. An analyst might wish to find the circumstances under which a pattern holds for an itemset and then find out which *other* sets satisfy the pattern under those circumstances. (Query Q_3 in Section 1 is an example involving such exploration.) Or she might start with a circumstance (say, fall in northeast), find the itemsets that satisfy a pattern, and then attempt to characterize the circumstances under which these latter sets satisfy the pattern (e.g., see query Q_5 in Section 1). Queries of this kind require that the analyst be able to freely move between the worlds of itemsets and circumstances. The bilattice of itemsets and circumstances developed in Section 3 is ideal for supporting such "migration." In this section, we propose the notion of an *Armstrong Basis* as a basis for supporting a variety of queries involving such migration. Recall the bilattice $(\mathcal{C} \times \mathcal{I}, \leq^{\rm up}_{\rm up}, \leq^{\rm up}_{\rm down})$ defined in Section 3. An element (ψ, S) in the bilattice is $\leq^{\rm up}_{\rm up}$-*minimal* w.r.t. a property p provided it satisfies p and whenever (ϕ, T) satisfies p and $(\phi, T) \leq^{\rm up}_{\rm up} (\psi, S)$, we have $(\psi, S) = (\phi, T)$. Minimality w.r.t. the other order and maximality w.r.t. either order is defined analogously.

Definition 1 (Armstrong Basis) Let p be a pattern over circumstances and itemsets, and $\mathcal{D}$ be a transaction database. Suppose p is c-monotone and i-anti-monotone. Then the *Armstrong Basis* of p w.r.t. $\mathcal{D}$ is defined as $\mathcal{AB}(p_{\mathcal{D}}) = \{(\psi, S) \in (\mathcal{C} \times \mathcal{I}) \mid (\psi, S) \text{ is } \leq^{\rm up}_{\rm down} - \text{minimal w.r.t. } p\}$.

[8] Note that the time scale is different in Figures B and C.

The significance of Armstrong Basis will be clear from the following proposition.

Proposition 2 *Let D a transaction database, p be a c-monotone and i-anti-monotone pattern, and $\mathcal{AB}(p_{\mathcal{D}})$ the Armstrong Basis of p w.r.t. D. Then the following statements are true.*

1. *For every element $(\psi, S) \in \mathcal{AB}(p_{\mathcal{D}})$, pattern p does not hold for any proper superset of S at circumstance ψ; similarly, p does not hold for S at any circumstance that is strictly stronger (i.e. it implies, but is not equivalent to, ψ) than ψ.*
2. *Every element of the bilattice $(\mathcal{C} \times \mathcal{I})$ which satisfies the above statement belongs to $\mathcal{AB}(p_{\mathcal{D}})$.*
3. *For every element (ϕ, T) in the bilattice, $p_D(\phi, T)$ holds if and only if there is an element $(\psi, S) \in \mathcal{AB}(p_{\mathcal{D}})$, such that $T \subseteq S$, and ϕ is logically implied by ψ.*

The proposition above also explains the reason behind the terminology used for this notion. Indeed, Armstrong Basis has a structure that is very similar to the notion of Armstrong couples introduced for families of functional dependencies by Armstrong, almost three decades ago [W74].

The notion of Armstrong Basis is applicable to any pattern that satisfies some monotonicity properties w.r.t. itemsets and circumstances, either by defining minimality or maximality w.r.t. the appropriate order in the bilattice. E.g., for *minpurchase*, a c-monotone and i-monotone pattern, the Armstrong Basis should be defined as the set of $\leq^{\mathrm{up}}_{\mathrm{up}}$-minimal elements satisfying *minpurchase*. Note that for such a pattern the Armstrong Basis has the complete information about the space of circumstance-itemset pairs valid w.r.t. this pattern, as stated in part (3) of the proposition.

In the rest of this section, we focus on algorithms for computing the Armstrong Basis. For a typical transaction database and a pattern such as frequency, the size of the Armstrong Basis can be substantial. So why should anyone want to compute it? The analogy we give is cube. For large databases, cube is a time- and space-expensive operation. Indeed, a user may not be interested in seeing a cube in its entirety. Its utility comes from its ability to service a variety of aggregation queries involving multiple group-bys. In a similar manner, if the Armstrong Basis for a pattern is available, queries (such as those in Section 1) involving migration between itemsets and circumstances can be answered efficiently. Thus, it is worth investigating efficient algorithms for its computation. We start with the following notions, defined for patterns p that are c-monotone and i-anti-monotone. Similar notions exist for other patterns.

Let p be a c-monotone and i-anti-monotone pattern. Define a function $p_i : \mathcal{C} \rightarrow \mathcal{I}$ as $p_i(\psi) = \{S \mid p(\psi, S) \wedge \forall S' : S' \supset S \Rightarrow \neg p(\psi, S')\}$, and a function $p_c : \mathcal{I} \rightarrow \mathcal{C}$ as $p_c(S) = \{\psi \mid p(\psi, S) \wedge \forall \phi : \phi < \psi \Rightarrow \neg p(\phi, S)\}$. These functions have obvious extensions to collections of circumstances or sets: for a set of circumstances Ψ, $p_i(\Psi) = \bigcup_{\psi \in \Psi} p_i(\psi)$, and for a collection of itemsets $\mathbf{S}$, $p_c(\mathbf{S}) = \bigcup_{S \in \mathbf{S}} p_c(S)$. Note that with such extensions, the functions p_i and p_c can be composed. For example, let p be frequency. Then for a collection of circumstances Ψ, the composite function $p_c \circ p_i$ does the following: first, it finds all maximal itemsets frequent under any circumstance in Ψ; second for each such itemset, it finds all minimal circumstances under which it is frequent. Similar remarks hold for other patterns. We say a set of circumstances Ψ *covers* another set Φ provided $\forall \phi \in \Phi, \exists \psi \in \Psi$, such that ψ implies ϕ. Similarly, a collection of itemsets $\mathbf{S}$ *covers* collection $\mathbf{T}$ provided $\forall T \in \mathbf{T}$, $\exists S \in \mathbf{S}$, such that $T \subseteq S$. Notice that the Armstrong Basis $\mathcal{AB}(p_{\mathcal{D}})$ is itself a cover of the set of all pairs (ψ, S) for which the pattern p holds. We have the following result.

Algorithm ArmBase;
Input: a transaction database D, a c-monotone and i-anti-monotone pattern p.
Output: $\mathcal{AB}(p_{\mathcal{D}})$

1. the Armstrong Basis is initially empty;
2. do a level wise sweep of the itemset lattice and do the following:
3. for each set S at the current level, if S is not a subset of S' for some (ψ, S') in the current Armstrong Basis, then find the collection of minimal circumstances $circ_S$ at which the pattern holds for S;
4. for each circumstance $\psi \in circ_S$, find the collection of maximal sets $itsets_\psi$ for which the pattern holds under ψ; let ITSETS be the union of $itsets_\psi$, over all $\psi \in circs_S$;
5. each pair in $circ_S \times itsets$ is an element of $\mathcal{AB}(p_{\mathcal{D}})$;
6. terminate when all levels are processed, or when no itemset at a given level makes it to $\mathcal{AB}(p_{\mathcal{D}})$;

Figure 6. Algorithm for computing the Armstrong Basis of a pattern in a transaction database

Proposition 3 *Let p_i and p_c be functions as defined above, w.r.t. a c-monotone and i-anti-monotone pattern p. Then the following holds:*

1. *for every collection of circumstances Ψ, $p_c(p_i(\Psi))$ covers Ψ; furthermore, $p_c(p_i(p_c(p_i(\Psi)))) = p_c(p_i(\Psi))$.*
2. *for every collection of itemsets $\mathbf{S}$, $p_i(p_c(\mathbf{S}))$ covers $\mathbf{S}$; furthermore, $p_i(p_c(p_i(p_c(\mathbf{S})))) = p_i(p_c(\mathbf{S}))$.*

The proposition can be proved by appealing to the definitions of p_i and p_c. It immediately suggests the naive algorithm in Figure 6 for computing $\mathcal{AB}(p_{\mathcal{D}})$.

This algorithm relies on the following facts: $circs_S = p_c(S)$ and $itsets = p_i(p_c(S)$. For each pair $(\phi, T) \in (p_c(S) \times p_i(p_c(S)$, if p held for T at a stronger circumstance, then it would hold there for S too, which contradicts the minimality of ϕ w.r.t. S. Similarly, if p held for a proper superset of T at ϕ, this would contradict the maximality of T w.r.t. circumstances in $p_c(S)$. The correctness follows from this. Unfortunately, this algorithm can be prohibitively expensive. In particular, it does not share any effort between successive iterations. There is, of course, a dual algorithm which works off the circumstance lattice, sweeping it levelwise, and is very expensive as well. In the full paper, we develop efficient algorithms for computing the Armstrong Basis, suppressed here for lack of space.

7 Discussion and Conclusions

Recently, there has been interest in mining long patterns. In this context, algorithms for mining maximal frequent sets have been proposed [Bay98, AAP00]. This is relevant because of our interest in minimal circumstances. However, these algorithms operate in a different lattice and the mechanisms of the algorithms appear different. Given our interest in the Armstrong Basis, these algorithms might offer yet another way to efficiently find maximal frequent itemsets associated with a circumstance, in that context. The recently proposed FP-tree method [HPY00, PHM00] has obvious relevance for circumstance mining and also for the computation of the Armstrong Basis. We are currently exploring efficient algorithms for the Armstrong Basis based on the FP-tree method. Finally, as previously noted, mining for circumstances in which a given pattern (like frequent itemset) holds has a close relationship with cube computation. For lack of space, we refer the reader to the recent paper [BR99] which surveys many of the cube methods, and which also describes the BUC algorithm.

In conclusion, basing pattern mining on a clear separation between item attributes and circumstance attributes pays its dividends: the close ties with the cube computation paradigm, and between cube computation and mining of i-anti-monotone patterns open up new and promising directions for future work. As motivated earlier, finding minimal (for c-monotone patterns) circumstances under which a pattern holds for a specific itemset is useful in its own right. The framework developed in this paper goes far beyond just support-based mining of frequent itemsets. This is achieved using properties of patterns on the bilattice of circumstances and itemsets. Finally, we proposed a useful notion of Armstrong Basis with which we can find (say, minimal) circumstances where a pattern holds for an itemset, and then find which other itemsets satisfy the pattern under these circumstances. This is in the spirit of mining the result of mining, one of the important problems identified by Imielinski and Mannila in their seminal paper [ImMa96]. Just as data cube is an important operator for supporting queries involving multiple group-bys, for queries involving migration between itemsets and circumstances, Armstrong Basis can provide effective support. More work is needed for its efficient computation and its use in answering such mining queries.

References

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *(VLDB'94)*, pp. 487–499

[AS95] R. Agrawal and R. Srikant. Mining sequential patterns. *(ICDE'95)*, pp. 3–14

[AAP00] R. Agrawal, C. C. Aggarwal, and V. V. V. Prasad. Depth first generation of long patterns. KDD 2000, to appear.

[Bay98] R. J. Bayardo. Efficiently mining long patterns from databases. *(SIGMOD'98)*, pp. 85–93

[BR99] K. S. Beyer, R. Ramakrishnan: Bottom-Up Computation of Sparse and Iceberg CUBEs. *SIGMOD '99*, pp. 359–370

[BMS97] S. Brin, R. Motwani, and C. Silverstein. Beyond market basket: Generalizing association rules to correlations. *SIGMOD '97*, pp 265–276

[DL99] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *KDD'99*, pp. 43–52

[Fit91] M. Fitting. Bilattices and the semantics of logic programming. *J. Logic Programming* **11**, 91–116.

[GLW00] G. Grahne, L. V. S. Lakshmanan, X. Wang. Efficient Mining of Constrained Correlated Sets. *ICDE 2000*, pp. 512–521,

[G*96] J. Gray, A. Bosworth, A. Layman, H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. *ICDE 1996*, pp. 152–159.

[HPY00] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *SIGMOD 2000*, pp. 1–12

[ImMa96] T. Imielinski, H. Mannila: A Database Perspective on Knowledge Discovery. CACM 39(11): 58-64 (1996).

[LNHP99] L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *SIGMOD'99*, pp. 157–168

[LSW97] B. Lent, A. Swami, and J. Widom. Clustering association rules. In *Proc. ICDE'97*, pp. 220–231

[MTV94] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In *KDD'94*, pp. 181–192

[MTV97] H. Mannila, H Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[NLHP98] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *SIGMOD'98*, pp. 13–24

[PHM00] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *DMKD 2000)*, pp. 11–20

[SVA97] R. Srikant, Q. Vu, and R. Agrawal. Mining association rules with item constraints. In *KDD'97*, pp. 67–73

[W74] W. W. Armstrong: Dependency Structures of Data Base Relationships. IFIP Congress 1974: 580-583.

[ZaHs99] M.J. Zaki and C. Hsiao. Charm: an efficient algorithm for closed association rule mining. Tech. Report., RPI, 1999.

[ZDN97] Y. Zhao, P. Deshpande, J. F. Naughton: An Array-Based Algorithm for Simultaneous Multidimensional Aggregates. *SIGMOD'97*, pp. 159–170.

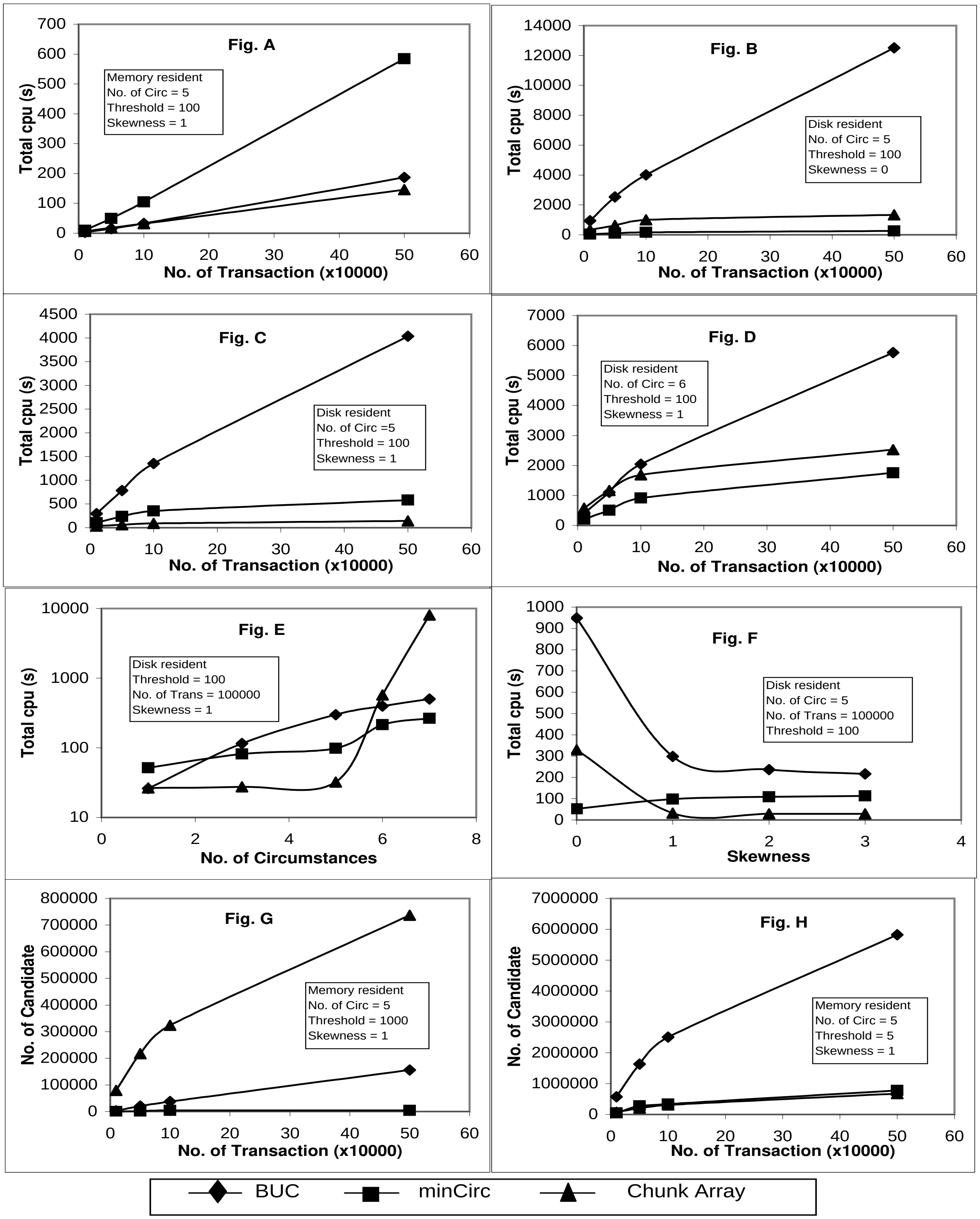

Figure 7. Performance of Three Algorithms for Computing Minimal Circumstances