INFORMS Journal on Computing
Articles in Advance, pp. 1–14
ISSN 1091-9856 (print) | ISSN 1526-5528 (online)

http://dx.doi.org/10.1287/ijoc.2013.0558

A Tree-Based Contrast Set-Mining Approach to Detecting Group Differences

Hongyan Liu
Research Center for Contemporary Management, Key Research Institute of Humanities and Social Sciences at Universities, School of Economics and Management, Tsinghua University, Beijing, China, 100084, liuhy@sem.tsinghua.edu.cn

Yinghui (Catherine) Yang
Graduate School of Management, University of California, Davis, California 95616, yiyang@ucdavis.edu

Zhuohua Chen, Yong Zheng
School of Economics and Management, Tsinghua University, Beijing, China, 100084
{chenzhh3.12@sem.tsinghua.edu.cn, zhengy3.06@sem.tsinghua.edu.cn}

Understanding differences between groups in a data set is one of the fundamental tasks in data analysis. As relevant applications accumulate, data-mining methods have been developed to specifically address the problem of group difference detection. Contrast set mining discovers group differences in the form of conjunction of feature-value pairs or items. In this paper, we incorporate absolute difference, relative difference, and statistical significance in our definition of a group difference, and develop a novel method named DIFF that uses the prefix-tree structure to compress the search space, follows a tree traversal procedure to discover the complete set of significant group differences, and employs efficient pruning strategies to expedite the search process. We conducted comprehensive experiments to compare our method with existing methods on completeness of results, pruning efficiency, and computational efficiency. The experiments demonstrate that our method guarantees completeness of results and achieves higher pruning efficiency and computational efficiency compared to STUCCO. In addition, our definition of group difference is more general than STUCCO. Our method is more effective than traditional approaches, such as classification trees, in discovering the complete set of significant group differences.

Key words: data mining; group difference detection; contrast set mining
History: Accepted by Alexander Tuzhilin, Area Editor for Knowledge and Data Management; received August 2010; revised August 2011, September 2012, January 2013, February 2013; accepted March 2013. Published online in *Articles in Advance*.

1. Introduction

It is common in both academic research and industry practice to analyze differences between groups of subjects (e.g., customers, patients, etc.). Finding such differences can help us better understand characteristics that distinguish these groups, from which we can make proper decisions and derive meaningful business strategies. Traditional statistical methods focus more on testing hypotheses, which are normally generated according to prior knowledge or theory. In this case, testing is often conducted to check whether two known groups are different along a known dimension. For example, "Will customers with different education levels react differently to a marketing campaign?" On the other hand, data-mining methods focus on the endogenous discovery of significant differences (of a certain format), considering all available dimensions and interactions between those dimensions. Data-mining methods that are specifically designed to uncover group differences include contrast set mining (Bay and Pazzani 1999) and emerging pattern mining (Dong and Li 1999). They each have a slightly different group difference definition, thus leading to different mining methods.

Bay and Pazzani (1999) coined the term "contrast set mining" and proposed the STUCCO (search and testing for understandable consistent contrasts) method to discover contrast sets. Contrast set mining seeks conjunctions of feature-value pairs or items, called contrast sets, which have significantly different levels of support in different groups. For example, if we are comparing education groups, we may discover that P (occupation = sales | Ph.D.) = 2.7%, while P (occupation = sales | Bachelor) = 15.8%. Then, (occupation = sales) is a contrast set between the group with a Ph.D. and the group with a bachelor's degree (given that the spread between 2.7% and 15.8% exceeds the given threshold). Contrast set-mining techniques focus on using heuristics to expedite the search for all such contrast sets. A feature-value pair

takes the form of [*feature op value*], where *op* can be $=$, $>$, $<$, $\geq$, or $\leq$. An example of a contrast set for online activity data is {amount of money spent $>$ \$100}, or {online time per day ≤ 1 hour, number of websites per session $= 3$}. A contrast set can also be a set of items (e.g., products or websites). An example contrast set of this format is {milk, egg, bread}.

Since the introduction of contrast set mining, it has been used in numerous applications (e.g., Kralj et al. 2007a, b), and it has also been extended methodologically (e.g., Loekito and Bailey 2008, Wong and Tseng 2005). A majority of the methodological improvements focus on exploring a more specific pattern format, such as $(\neg X) \wedge Y$ instead of $X \wedge Y$ (Wong and Tseng 2005), or discovering a specific subset of contrast sets, such as the "interesting" ones (Loekito and Bailey 2008). However, there has not been any significant improvement in the core method of STUCCO. STUCCO will discover the complete set of contrast sets that satisfy a given condition (e.g., $|P(contrastset \mid group1) - P(contrastset \mid group2)| > \delta$). In this paper, the format of our group difference is also conjunctions of feature-value pairs or items. However, our definition is more general compared to that of STUCCO. On top of the absolute difference, we also incorporate the relative difference in our definition of a group difference. We propose a novel group difference detection algorithm named DIFF that uses the prefix-tree structure to compress the search space, follows a tree traversal procedure to discover the complete set of group differences, and employs efficient pruning strategies to expedite the search process. We conduct comprehensive experiments to compare our method with STUCCO based on completeness of results, pruning efficiency, and computational efficiency. This paper demonstrates that DIFF guarantees completeness of results and achieves higher pruning efficiency and computational efficiency than STUCCO.

Some of these differences can also be discovered by classification models. However, the goal of classification models is to maximize the overall predictive accuracy; thus, the model learning process does not guarantee the discovery of all of the most significant group differences. Take the decision tree model as an example: if we treat the group assignment as the dependent variable, decision tree models can be used to learn the features that lead to different group assignments (which group an instance should belong to). By following a branch of the tree, we can obtain a conjunction of feature-value pairs, which can be a potential group difference if its frequency of occurrence in each group is significantly different. However, we might miss some very significant group differences by using decision tree methods. Because decision-tree building process often follows a greedy rule for each split; it is not guaranteed that every significant group difference will be discovered by a specific tree. In the online supplement (available as supplemental material at http://dx.doi.org/10.1287/ijoc.2013.0558), we use a detailed example to further illustrate this point. As long as the method does not consider the entire search space of significant group differences, there are possibilities of missing some important ones. When the goal is not to classify or predict, using classification-based methods (e.g., decision trees, and feature selection for classification) for group difference detection purpose will likely generate inferior results. Therefore, we need to design tailored methods to find significant group differences.

Discovering such group differences has a wide range of applications. As we can see, the components (or items) of a contrast set can be feature-value pairs (e.g., age $>$ 45, income $=$ high) and simple items/products (e.g., long-term service plan, history book). The flexibility of the group difference representation makes it very powerful in capturing many hidden connections.

A very natural application for contrast set mining is marketing (Song et al. 2001), because of the availability of large volumes of customer and transaction data. Three illustrative examples of contrast sets in the field of marketing are as follows.

(1) {gender $=$ male, income $=$ high} with respect to spending on electronics. A finding that P((gender $=$ male) $\wedge$ (income $=$ high) | high spending on electronics) is significantly higher than P((gender $=$ male) $\wedge$ (income $=$ high) | low spending on electronics)) indicates that male customers with high income tend to spend a lot on electronics. This pattern helps managers design more effective targeted marketing campaigns.

(2) {history book, history book} with respect to adopting a recommender system. The finding that P((history book) $\wedge$ (history book) | after recommender system implementation) is significantly higher than P((history book) $\wedge$ (history book) | prior to recommender system implementation) indicates that customers made multiple purchases in the history category after the recommender system went into effect. This could be valuable information for managers to evaluate the recommender system. Here, the item is a book category instead of a specific book title, but the same logic applies to categories and individual items.

(3) There are many useful group differences that can be discovered from millions of items (e.g., products, websites, words, etc.). For example, a store has conducted a promotion campaign and would like to see how things compare before and after the promotion. One of the interesting differences discovered is {electronic toothbrush, electronic toothbrush heads}.

After the promotion, this item set occurs much more frequently in the sales data observed over a period of time. A study of this group difference could indicate that the coupons printed after customers check out are working effectively. A further study during a longer period of time following the discovery of this difference could reveal that promotions can, to some degree, modify customers' consumption behavior (e.g., they change their electronic toothbrush heads more often). Without an effective group difference detection method, it will be hard for managers to pinpoint this difference among millions of sales records, and they will have difficulties knowing where to look to make better decisions.

In addition to marketing, the field of finance may also benefit from contrast set mining, because there are a large number of factors that can influence the performance of a company or a stock. It is also very popular in finance to perform event analysis where data are compared before and after a certain event. Finding group differences can help nail down the hidden connections between various factors. For example, finding differences between good performing stocks and bad performing ones can give some guidance on stock selections. A contrast set might be $\{P/E$ ratio < 10, industry $=$ IT, market cap $<$ \$1 billion$\}$ when comparing stocks with $> 5\%$ yield in the past month to those with $\leq 5\%$ yield. With this finding, investors can appropriately focus on smaller IT companies with a relatively low P/E ratio (price-earnings ratio) if they are seeking fast-growing stocks. When comparing data before and after the quantitative easing in 2008, we may find that {food industry $=$ low growth, IT industry $=$ high growth} exhibits a contrast set indicating that the quantitative easing policy negatively affected the food industry and positively affected the IT industry. In the future, when such policies are considered, the influence of industry changes may also need to be considered.

Aside from business applications, group difference detection has many other applications in social science, bioinformatics, medical research, etc. For example, Nazeri et al. (2008) discovered differences between aircrafts that had been involved in accidents and those that had not. They used these differences to uncover several important causes of airplane accidents. In medical practice, patients can be divided into groups according to certain features (e.g., age, gender, disease), and group difference detection can uncover key symptom differences across groups. This technique is also used to study the effectiveness of medicines or treatments by comparing differences between patients who are undergoing medical treatments (Kralj et al. 2007a, b). In social science research, it is even more common to compare differences between different groups of populations. Ruggles (1997) and Darity (2000) used census data to analyze population disparity correlated with race, ethnicity, education, and marital status. Minaei-Bidgoli et al. (2004) discovered contrasting rules describing interesting characteristics of performance disparity between various groups of students in the context of Web-based educational systems, and used these differences to evaluate the effectiveness of a number of learning strategies.

The remainder of the paper is organized as follows: We survey related work in §2. Section 3 defines the problem and introduces the definition of group differences. Section 4 presents the prefix tree-based structure to compress data and the tree traversal procedure to discover all group differences as well as the pruning strategies to expedite the search process. Section 5 reports the experimental results comparing our method and the popular STUCCO method. Finally, §6 summarizes our study and outlines an agenda for future research.

2. Related Work

In this section, we review methods that can be used to discover differences between groups. Among these methods, contrast set-mining and emerging pattern-mining methods are specifically designed to find group differences. Thus, these two methods are discussed in detail here. We also survey other methods that are not tailored for this problem, but instead can be adopted for this purpose (these methods may not return the best results).

Contrast Set Mining. Contrast set mining was first introduced in Bay and Pazzani (1999) to discover group differences. A contrast set is defined as an item set or conjunction of feature-value pairs whose support differs significantly across groups, i.e., $\max_{ij}|\text{support}(\text{cset}, G_i) - \text{support}(\text{cset}, G_j)| \geq \delta$, where G_i and G_j are any two groups among all of the predefined groups, δ is a user-defined threshold for the minimum support difference, and the support of an item set is defined to be the percentage of transactions in a group containing that item set. The search algorithm STUCCO and accompanying pruning rules were introduced to find all such contrast sets.

Since the introduction of contrast set mining, it has been used in numerous applications and extended methodologically. A majority of the methodological improvements focus on exploring a specific pattern format (e.g., $(\neg X) \wedge Y$ instead of $X \wedge Y$) or discovering a specific subset of contrast sets (e.g., the "interesting" ones). However, there has not been significant improvement in the core method of STUCCO. Wong and Tseng (2005) proposed a method to discover negative contrast sets (i.e., $(\neg X) \wedge Y$ instead of $X \wedge Y$) across groups, extending contrast sets to a different format. An example of a

negative contrast set is {occupation $<>$ accounting, gender = male} where P(occupation $<>$ accounting, gender = male | Ph.D.) is found to be significantly different from P(occupation $<>$ accounting, gender = male | Bachelor). Although Wong and Tseng (2005) did not provide an improvement directly over STUCCO, they expanded the concept of a contrast set to include a negative contrast set. Hilderman and Peckham (2007) compared STUCCO with their own method, which relies on different statistical philosophies than STUCCO and finds a different set of group differences, but the paper demonstrated that both methods are statistically sound. Kralj et al. (2007a, b) performed contrast set mining through subgroup discovery (Lavrac et al. 2004). Satsangi and Zaiane (2007) use the association rule-based approach to find contrast sets. At the same time, STUCCO has been used to address real-world problems in many fields, such as education (Minaei-Bidgoli et al. 2004), medical research (Kralj et al. 2007a, b; Siu et al. 2005), and the aviation industry (Nazeri et al. 2008). Aside from the research extending contrast sets to a more specific format, there is other research that looks at finding interesting contrast sets among all discovered contrast sets (Loekito and Bailey 2008). None of the previous methods achieved a significant improvement over STUCCO's core method. In this paper, we provide a more general definition for a contrast set incorporating both absolute and relative differences. In addition, we provide a much more efficient method to find the significant group differences.

Emerging Pattern Discovery. Another concept that is closely related to contrast sets is emerging patterns, which describe significant changes between two classes of data. This approach was first introduced by Dong and Li (1999) and developed in parallel with contrast set mining. Both contrast sets and emerging patterns can be used to detect differences between groups. Emerging patterns are also defined as item sets (or conjunctions of feature-values pairs) whose support increases significantly from one data set to another. This technique is often used to capture emerging trends in time-stamped databases, or to capture differentiating characteristics between classes of data. After the introduction of emerging patterns, subsequent emerging-pattern research has largely focused on the use of the discovered patterns for classification purposes instead of proposing better methods to discover such patterns (Ramamohanarao et al. 2005). For example, Ramamohanarao (2010) studied classification by emerging patterns, and Kobyliński and Walczak (2011) studied classification by jumping emerging patterns, which are defined as emerging patterns that have zero support in at least one group. Fan and Ramamohanara (2003) proposed a Bayesian approach based on emerging patterns, and bagging was proposed by Fan et al. (2006). Loekito and Bailey (2006) designed more efficient algorithms for discovering emerging patterns and extended the technique to more complex pattern formats.

The main difference between emerging patterns and contrast sets is that contrast sets measure an absolute difference, and emerging patterns measure a relative difference (i.e., P(item set | group 1) $-$ P(item set | group 2) versus P(item set | group 1)/P(item set | group 2)). Because there is no lower support threshold for an emerging pattern, it is extremely challenging to find all of the emerging patterns. For example, the value 0.001%/0.0001% is a very high ratio despite the absolute support value being very low. If an item set (as a group difference) appears once in one group and twice in the other group (assuming these two groups are of similar size), then it is a significant group difference according to relative difference, but it is a very trivial one. And there will be a massive number of such item sets, leading to great computational expense. Absolute difference helps us filter out these types of item sets.

There is also other research that studies changes/differences between data sets, which does not fall into the prior categories. Liu et al. (2001) detects the changes in support and confidence of association rules (instead of item sets) from different groups. Deng and Zaïane (2009) find subsequences that are frequent in sequences of one group and less frequent in the sequences of another, thus distinguishing or contrasting sequences of different classes. Liu et al. (2000) and Wang et al. (2003) present techniques that identify differences in the decision trees and classification rules, respectively, found on two different data sets. Alqadah and Bhatnagar (2009) design an unsupervised method and consider differences in both the attribute and object space.

Aside from methods that are specifically designed for mining group differences, other more general methods that are normally used for other purposes can be modified to apply to group difference discovery. Decision-tree induction (Breiman et al. 1984, Quinlan 1993) and association rule mining (Agrawal and Srikant 1994, Webb 2000) are two such methods. As discussed in §1, a branch of a decision tree can correspond to a potential group difference. However, decision tree models may generate suboptimal results (see the online supplement for more detailed discussion on this topic). Webb et al. (2003) applied an association rule mining system, Magnum Opus (Webb 2001), to discover differences across groups. A group difference can be treated as a special type of association rule with the set of items on the left-hand side of the rule and the group identity on the right-hand side of the rule. Most of the rules discovered by association rule mining methods are not the type of rules that represent group differences. Thus, this approach involves unnecessary computation for our purpose.

3. Group Difference Definition

We consider a transaction database that contains a collection of transactions. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items or feature-value pairs, and a transaction $T_{il} \subseteq I$. Each transaction belongs to a group G_i, where $G_i \in G = \{G_1, G_2, \ldots, G_n\}$ and $G_i \cap G_k = \varnothing$ (for $i, k \in \{1, 2, \ldots, n\}$). Each transaction also belongs to a transaction generator u_{ij}, where $i = \{1, 2, \ldots, n\}$ indicates that u_{ij} belongs to group G_i, and $j = \{1, 2, \ldots, m_i\}$ where m_i is the total number of unique transaction generators in group G_i. The transaction generator can be interpreted as a customer, account, Web user, etc. Our method is not restricted to data with specified transaction generators. When transaction generators are not specified, it can be interpreted as a special case in which each transaction comes from a different transaction generator. Incorporating transaction generators into our framework gives decision makers more flexibility when they sense that unbalanced transaction distribution across different groups may introduce biases. For simplicity, we use "user" to refer to a transaction generator in the rest of the paper. An item in a transaction can be an actual item in the store (e.g., an apple), or a feature-value pair (e.g., gender = male). Given this definition, an illustrative set of transactions in group G_i is represented in Table 1.

Each row in this table corresponds to a transaction, and a user can have multiple transactions. The size of each transaction (i.e., the number of items contained in a transaction) can be different. In the data-mining literature, an item set of size k is defined as a set containing k unique items $\{i_1, i_2, \ldots, i_k\}$. According to the traditional definition (Agrawal and Srikant 1994), the support of an item set within a group is the percentage of transactions containing the item set (i.e., all the items in the item set appear in the transaction). This definition does not consider the number of transactions each user has. If a certain user has a large number of transactions, this user will have a strong influence on the support calculation under the traditional support definition. In this paper, we adopt a more general definition of support that considers transaction generators. The traditional support definition becomes a special case of our definition. The decision on whether transaction generators should be incorporated is often a managerial one. Under the circumstances where managers prefer the transaction generators to have relatively similar weights no matter how many transactions each generator generated, we can incorporate the transaction generator in the support calculation. Under the cases where managers think that each transaction should have similar importance (e.g., purchase transactions), we can simply assign each transaction a unique transaction generator; then our definition simply becomes the traditional support definition. Our support definition accommodates both cases, giving managers more flexibility.

Table 1 **Transactions in Group G_i**

Transaction ID	Transaction generator	Set of items (each item $\in I$)
T_{i1}	u_{i1}	$\{i_{11}^{G_i}, i_{12}^{G_i}, \ldots, i_{1a_1}^{G_i}\}$
T_{i2}	u_{i1}	$\{i_{21}^{G_i}, i_{22}^{G_i}, \ldots, i_{2a_2}^{G_i}\}$
$\cdots$	$\cdots$	$\cdots$
$T_{i(N_i-1)}$	$u_{i(m_i)}$	$\{i_{(N_i-1)1}^{G_i}, i_{(N_i-1)2}^{G_i}, \ldots, i_{(N_i-1)a_{(N_i-1)}}^{G_i}\}$
$T_{i(N_i)}$	$u_{i(m_i)}$	$\{i_{N_i1}^{G_i}, i_{N_i2}^{G_i}, \ldots, i_{N_ia_{N_i}}^{G_i}\}$

Notes. Each item symbol in the third column refers to an actual item from I, which is $\{i_1, i_2, \ldots, i_m\}$. A transaction contains a list of items from I, and the number of items in each transaction does not need to be the same.

Definition 1 (Support of an Item Set Within a Group). The support of an item set l within group G_i is defined as the average of the support of this item set across all users in that group.

$$\text{Support}(l \mid G_i) = \frac{\sum_{j=1}^{m_i} \text{Support}(l \mid (u_{ij}, G_i))}{m_i},$$

where

$$\text{Support}(l \mid u_{ij}, G_i) = \frac{\text{num\_of\_trans}(l \mid u_{ij}, G_i)}{\text{num\_of\_trans}(u_{ij}, G_i)}.$$

Similarly, the count of the item set l within group G_i is defined as count = support · num_of_users, which is

$$\text{count}(l \mid G_i) = \sum_{j=1}^{m_i} \frac{\text{num\_of\_trans}(l \mid u_{ij,} G_i)}{\text{num\_of\_trans}(u_{ij}, G_i)}.$$

num_of_trans(u_{ij}, G_i) is the number of transactions generated by u_{ij} in group G_i, and num_of_trans$(l \mid u_{ij} G_i)$ is the number of transactions containing item set l among the transactions generated by u_{ij} in group G_i. Please see the online supplement for an example illustrating the benefit of Definition 1.

Definition 2 (Group Difference). The group difference set L is a collection of item sets such that for each $l \in L$, we have:

$$\max(\text{Support}(l \mid G_1), \ldots, \text{Support}(l \mid G_n)) - \min(\text{Support}(l \mid G_1), \ldots, \text{Support}(l \mid G_n)) \geq \min\_\text{dif}; \quad \text{(M1)}$$

$$\frac{\min(\text{Support}(l \mid G_1), \ldots, \text{Support}(l \mid G_n))}{\max(\text{Support}(l \mid G_1), \ldots, \text{Support}(l \mid G_n))} \leq \min\_\text{ratio}; \quad \text{(M2)}$$

$$(\exists\, i, j)[P(l \mid G_i) \neq P(l \mid G_j)]. \quad \text{(M3)}$$

Here, min_dif is a user-specified threshold for absolute difference, and min_ratio is a user-specified threshold for min/max ratio, which measures relative difference (we can also use max/min or (max-min)/max to represent relative difference, but min/max is equivalent and easier to measure). Both absolute difference (M1) and relative difference (M2) are adopted so that the group difference definition can be more general. And, as we discuss more in the online supplement, incorporating relative difference will guarantee the completeness of the results, and adopting absolute difference will filter out a large number of trivial results. Thresholds min_dif and min_ratio can be configured to reflect real-world requirements regarding the importance of the absolute and relative difference. Criterion (M3) is to ensure that a group difference is statistically significant.

Theorem 1. *Condition* (M1) *is equivalent to the following condition* (M4).

$$\left|\text{Support}(l|G_i)-\text{Support}(l|G_j)\right| \geq \text{min\_dif}, \quad \text{for some } i,j\in\{1,2,\ldots,n\}, i\neq j. \quad \text{(M4)}$$

Proof. Please see the online supplement.

4. The Tree-Based Approach for Discovering Group Differences

The group difference detection problem is to find all group differences that satisfy (M2)–(M4). The space of all possible group differences is huge, and we need to design an efficient method to find all the significant group differences. For discovering all group differences that satisfy (M2)–(M4), we propose a novel group difference detection algorithm named DIFF that uses the prefix-tree structure to compress the search space, follows a tree-traversal procedure to discover the complete set of group differences, and employs efficient pruning strategies to expedite the search process. Our method returns the complete set of group differences that satisfy (M2)–(M4), and achieves high efficiency. In §§4.1–4.3, we will use an example data set to illustrate the details of the main components in our method, and in §4.4, we will elucidate the formal algorithm.

4.1. Creating the Search Space

We organize the search space into several prefix trees. All transactions in a group are condensed into a tree. By using trees we can significantly reduce search complexity, and the tree structure also allows us to further reduce the search space through efficient pruning.

We begin by selecting frequent items. An item is considered frequent if its frequency in at least one group exceeds the minimum frequency threshold (min_sup). This is different from the traditional definition of a frequent item, which only considers frequency in an entire data set. An item that is frequent only in one group will likely have a high group difference value, because of the variance in the item's frequency across groups.

Table 2 Sample Data Set

G_1		G_2		G_3	
User ID	Transaction	User ID	Transaction	User ID	Transaction
100	{*b*, *e*}	10	{*b*, *c*, *d*}	1	{*b*, *c*}
100	{*b*, *c*, *d*}	10	{*a*, *d*}	2	{*a*, *b*}
200	{*b*, *e*}	10	{*b*, *c*}	2	{*a*, *d*}
200	{*a*, *d*}	20	{*a*, *b*, *e*}	2	{*c*, *e*}
300	{*a*, *c*}	20	{*a*, *b*, *c*, *e*}	3	{*a*, *b*, *c*}
300	{*b*, *c*}	30	{*a*, *c*}	3	{*b*, *c*, *d*}
40	{*a*, *c*}	30	{*b*, *d*}		
400	{*a*, *b*, *c*}				
400	{*a*, *b*, *c*}				

An example that illustrates tree construction is as follows.

In Table 2, an entire data set is organized into three groups. Table 3 lists the support of each frequent item in each group. Given the support threshold min_sup = 12%, we derive the frequent items {*a*, *b*, *c*, *d*, *e*}. As we can see from Table 3, the support of *e* in group G_3 is lower than the support threshold.

We now rank the items according to their support value in group G_1 and generate three lists in Table 4. We also sort the items according to the support values in another group. What we need is a sorted list that we can use to simultaneously traverse all the trees in a predefined order (details are discussed later).

Once we have selected the set of frequent items (here they are {*a*, *b*, *c*, *d*, *e*}), we build a tree to store

Table 3 Support Values for Items {*a*, *b*, *c*, *d*, *e*}

Item	Support in G_1 (%)	Support in G_2 (%)	Support in G_3 (%)
a	(0 + 1/2 + 1/2 + 3/3)/4 = 50	(1/3 + 2/2 + 1/2)/3 = 61	(0 + 2/3 + 1/2)/3 = 39
b	67	72	78
c	63	56	78
d	25	39	28
e	25	33	11

Note. The support is calculated according to Definition 1.

Table 4 Support Values for Items {*a*, *b*, *c*, *d*, *e*} (Sorted)

Item	Support in G_1 (%)	Support in G_2 (%)	Support in G_3(%)
b	67	72	78
c	63	56	78
a	50	61	39
d	25	39	28
e	25	33	11

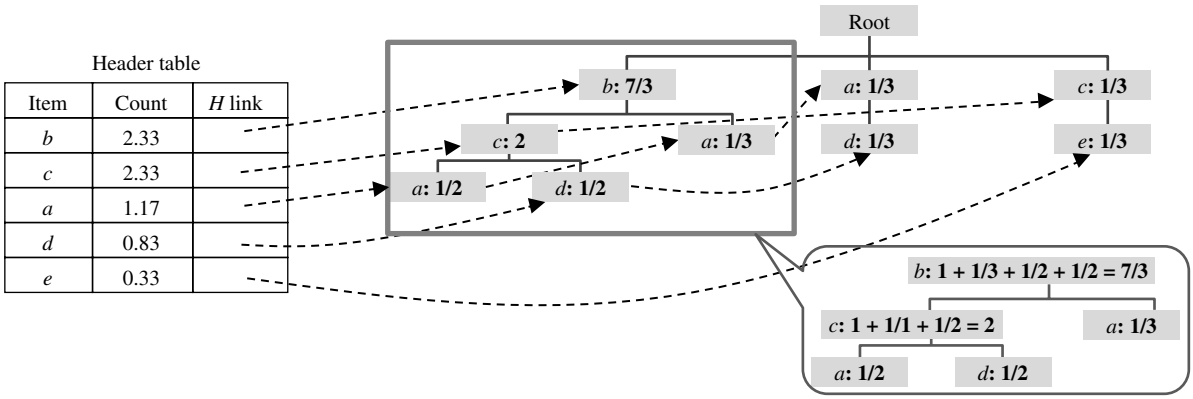


Figure 1 The Tree and Header Table for G_3

the transactions for each group. Take group G_3 as an example. We first create a header table for this group (see Figure 1). The first column is the frequent items sorted according to the predefined order. The second column shows the count (given in Definition 1) for each of the items in group G_3. The third column stores the link to nodes in the tree (discussed later). We first create the null root for the tree. For each transaction in G_3, we sort the items in the transaction according to the order in the header table and create a path in the tree following that order. For example, for the first transaction $\{b, c\}$ in G_3, the items are already sorted according to the predefined order. After we read in the first transaction from G_3, we add a path in the tree according to the order $\{b, c\}$. Thus, b becomes a child node of the root, and c is the child node of b. Within this transaction, the count of $\{b, c\}$ is $1/1 = 1$ (one transaction total from this user and one transaction within this user containing $\{b, c\}$), so we record this in both nodes $\{(b:1), (c:1)\}$. For the second transaction in G_3, $\{a, b\}$, we first resort to get $\{b, a\}$, and its count is $1/3$ (three transactions total from this user). We create a path for this transaction. Because b already exists in the tree, these two transactions can share the same b node, and we only need to add a child node $\{a\}$ to node $\{b\}$. The count of $\{b\}$ is updated to $1 + 1/3$, and the count of node $\{a\}$ is $1/3$. Similarly, when processing the rest of the transactions, we first check whether the existing tree has a common path. If there is a path that can be shared, we update the count of the nodes on the shared path, and then add new nodes following the shared path. Figure 1 presents the final tree together with the count values.

The header table is created to facilitate the tree traversal. The items in the header table are all the frequent items (frequent in at least one group), and they are sorted by the count or support value. The third column of the table stores the link pointing to the first occurrence of that item in the tree. Other nodes with the same item are linked together using node links based on the sequence of their appearances in the tree (going through the paths from left to right and for each path going from the leaf to the root). Using the header table, we can easily locate all the paths containing a certain item.

Following the tree construction process, we can easily reach the following theorem.

Theorem 2. *The support of a node is not greater than the support of its parent node.*

Proof. Please see the online supplement.

The prefix-tree structure we use in this paper is an extension based on the FP-tree structure (Han et al. 2004). Compared to Han et al. (2004), we consider multiple groups and multiple users; thus, the calculation of support incorporates users (Definition 1), and each group has its own tree. If there is only one group and one user, the tree built is the same as an FP tree. Similar to the proof of Lemma 2.1 in Han et al. (2004), we know that the complete set of transactions (including frequent items) in the database can be derived from the fully-constructed trees. Without referring to the original transaction database, we only need to traverse the trees to derive all the group differences. The tree is a highly compact structure storing all the necessary information, and the size of the tree is substantially smaller than the original database (Han et al. 2004).

4.2. Pruning the Search Space

Before we introduce the procedure for uncovering all of the group differences, we first present the pruning strategy. The principle of pruning is to reduce the search space when we foresee that item sets generated by certain branches cannot satisfy the condition for a group difference. Conditions (M2) and (M4) are used during this pruning process.

Let Ω be the set of all item sets that appear in at least one transaction in our database (and therefore in at least one group). Assume that I_k is the set of all item sets $l \in \Omega$ that contain the item set l_k (i.e., $\forall\, l \in I_k$, we

have $l_k \subseteq l$). The support of l_k in group G_i is the upper bound of the support of any item set in I_k. That is,

$$\text{UpperBound}(I_k \mid G_i) = \text{Support}(l_k \mid G_i), \quad i \in \{1, 2, \ldots, n\}.$$

Across all groups, we denote the maximum upper bound of the support of I_k as $\max \text{UpperBound}(I_k)$, where

$$\max \text{UpperBound}(I_k) = \max\{\text{UpperBound}(I_k \mid G_i)\}_{i \in \{1, 2, \ldots, n\}}.$$

Similarly, we can find the minimum support of the item sets in I_k across all groups. It is normally the support of the longest item set containing l_k. That is,

$$\text{LowerBound}(I_k \mid G_i) = \min\{\text{Support}(l \mid G_i), l \in I_k\}_{i \in \{1,2,\ldots,n\}}$$

and

$$\min \text{LowerBound}(I_k) = \min\{\text{LowerBound}(I_k \mid G_i)\}_{i \in \{1,2,\ldots,n\}}.$$

Unless every item set in I_k occurs in every group, the $\min \text{LowerBound}(I_k)$ is 0.

Property 1. *For any item set l_k, the set I_k contains all its supersets ($\forall l \in I_k, l_k \subseteq l$). The* BoundWidth *defined here is greater than the support difference between any two groups for any item set l in I_k, and the* BoundRatio *defined is smaller than the support ratio between any two groups for any item set l in I_k. That is,*

$$\text{BoundWidth}(I_k) = \max \text{UpperBound}(I_k) - \min \text{LowerBound}(I_k), \quad (5)$$

$$\text{BoundRatio}(I_k) = \min \text{LowerBound}(I_k) / \max \text{UpperBound}(I_k). \quad (6)$$

Based on Property 1, we propose the following pruning policy.

Pruning Policy. If the *BoundWidth* of I_k is smaller than the predefined difference threshold min_dif, or the *BoundRatio* of I_k is greater than the predefined ratio threshold min_ratio, then none of the item sets in I_k can satisfy both (M2) and (M4).

When the transactions are organized into trees, it is fairly easy to implement the pruning policy as illustrated in the example that follows. Figure 2 shows all three trees we built ("sup" in the figure means support).

Take $l_k = \{d\}$; then I_k will be the set of item sets containing $\{d\}$. We have $\text{UpperBound}(I_k \mid G_i) = \text{Support}(d \mid G_i)$. For example, in group G_1, the support of d is 1/4; thus, $\text{UpperBound}(I_k \mid G_1) = 1/4$. Following the node links, we can identify all the branches with d (nodes with dotted lines in Figure 2). The item sets corresponding to these branches are $\{b, d\}$ and $\{a, d\}$, both with support of 1/8 (the number in Figure 2 shows the count, so the support is $(1/2)/4 = 1/8$). Thus, $I_k = \{\{b, c, d\}, \{b, d\}, \{c, d\}, \{a, d\}\}$ for group G_1. Then, $\text{LowerBound}(I_k \mid G_1) = \sup(\{b, c, d\}) = \sup(\{a, d\}) = 1/8$. Similarly, we can calculate the upper bound and lower bound for other two groups.

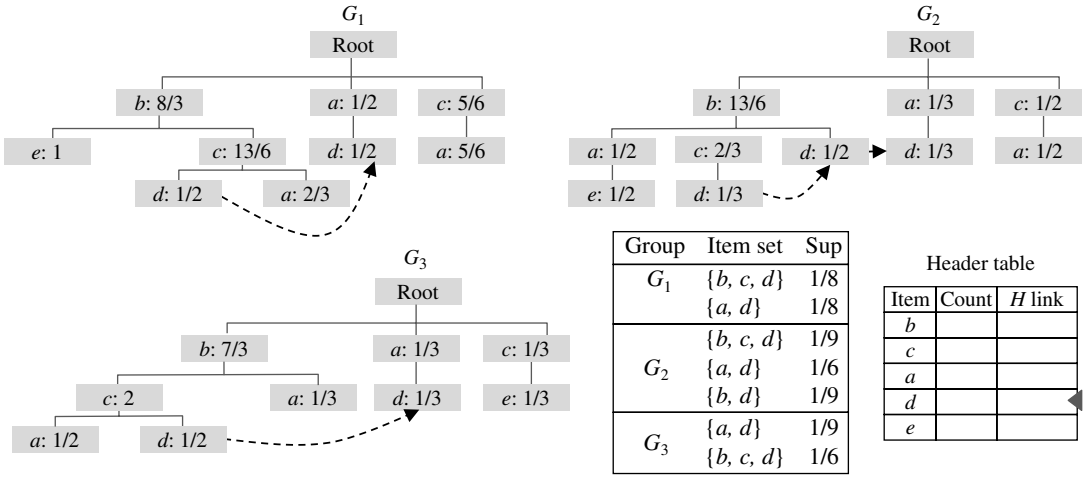
$$\text{UpperBound}(I_k \mid G_2) = \tfrac{7}{18}, \quad \text{LowerBound}(I_k \mid G_2) = \tfrac{1}{9};$$

$$\text{UpperBound}(I_k \mid G_3) = \tfrac{5}{18}, \quad \text{LowerBound}(I_k \mid G_3) = \tfrac{1}{9}.$$

Now we are able to compute the *BoundWidth* and *BoundRatio*.

$$\textit{BoundWidth}(I_k) = \max(\tfrac{1}{4}, \tfrac{7}{18}, \tfrac{5}{18}) - \min(\tfrac{1}{8}, \tfrac{1}{9}, \tfrac{1}{9}) = \tfrac{5}{18},$$

$$\textit{BoundRatio}(I_k) = \min(\tfrac{1}{8}, \tfrac{1}{9}, \tfrac{1}{9}) / \max(\tfrac{1}{4}, \tfrac{7}{18}, \tfrac{5}{18}) = \tfrac{2}{7}.$$



Group	Item set	Sup
G_1	$\{b, c, d\}$	1/8
	$\{a, d\}$	1/8
G_2	$\{b, c, d\}$	1/9
	$\{a, d\}$	1/6
	$\{b, d\}$	1/9
G_3	$\{a, d\}$	1/9
	$\{b, c, d\}$	1/6

Header table

Item	Count	*H* link
b		
c		
a		
d		
e		

Figure 2 Three Trees

If the difference threshold is 0.3, which is higher than 5/18, we can delete I_k from the search space, because 5/18 is less than 0.3. If the difference threshold is 0.2, but the ratio threshold is 0.25, we can also delete I_k from the search space. When implementing this in the tree, we can ignore the paths containing d (see the next section for more details.)

Next we show that our pruning strategy is superior to the set-enumeration tree-based pruning strategy used in Bay and Pazzani (1999, 2001). This comparison is based on the pruning by (M4) only, because Bay and Pazzani (1999, 2001) only incorporate absolute difference in its definition of group difference. Bay and Pazzani (1999, 2001) place nodes with the same parent in the set-enumeration tree (i.e., all nodes with a common prefix) into the same group. Within the group for a node l, they maintain two lists of items: the head $h(l)$, which is the common prefix for all sets in the group, and the tail $t(l)$ which is the set of all one-item extensions to the prefix. We merge the two sets $h(l)$ and $t(l)$ into $m(l)$, which is the largest possible item set containing l. In a data set with n groups $G_1, G_2, \ldots, G_n$, the pruning strategy in Bay and Pazzani (1999, 2001) is to prune all of the children nodes of l, if the following condition is satisfied:

$$\max[\text{support}(h(l)|G_i)]_{i\in\{1,2,\ldots,n\}} - \min[\text{support}(m(l)|G_i)]_{i\in\{1,2,\ldots,n\}} < \text{min\_dif}, \quad \text{(M4)}$$

where min_dif is the given group difference threshold.

Property 2. *Given the group difference threshold* min_dif *and an item set* l, *if* (M4) *is satisfied for* l, *then* BoundWidth$(l) <$ min_dif *is guaranteed to be satisfied. On the other hand, if* BoundWidth$(l) <$ min_dif *is satisfied for the node* l, (M4) *is not guaranteed to be satisfied for* l.

Proof. Please see the online supplement.

From Property 2, we can directly infer that our pruned set is a superset of that of STUCCO under the same group difference threshold min_dif.

4.3. Discovering Group Differences from the Trees

After creating a tree for each group, we need to traverse the trees simultaneously to discover all the group differences satisfying conditions (M2)–(M4). During the traversal process, we will also make pruning decisions. Again, we use the example based on the data set presented in Table 2 to further illustrate the process of discovering all the group differences based on the trees constructed earlier. For simplicity, we use only absolute difference (min_dif) in this example. The process for relative difference (min_ratio) works the same. We set min_sup to be 12% and min_dif to be 5% (min_ratio is assumed to be high enough). As shown in Figure 2, we have constructed three trees to represent the transactions in three groups, and the header table has the frequent items ranked in order (the other two columns in the header table will be different for each group.) We start from the bottom of the header table to first process all of the item sets containing e. Now, assume that we have finished processing e and moved to process d (after we consider the item sets containing item d, we will consider a, c, and b in turn.) First, through the node links, we locate all branches in the three trees containing d, and calculate the value of *BoundWidth* for all item sets containing d, and this value is 5/18 (28%), which does not satisfy the pruning condition (*BoundWidth* < min_dif). Therefore, we do not prune the branches with d. Instead, we check whether d is a group difference by calculating the support of d in all three groups, 25%, 38.89%, and 27.78%, which satisfy the min_dif threshold ($38.89\% - 25\% > 5\%$). Then item d becomes a qualifying group difference. After this, we will check whether any of the item sets containing d is a group difference. This step is similar to the FP-growth procedure proposed in Han et al. (2004). We know from the tree-building process that only items on d's prefix paths occurred together with d in the original database. These prefix paths, together with d's count value, form the conditional pattern base (i.e., the only patterns we need to consider conditional on d's existence). Table 5 lists the conditional pattern base and conditional trees for d.

The process we follow to construct the conditional trees of d from the conditional pattern base is similar to the process of building the original tree. We derive the set of frequent items according to the conditional pattern base of d, and we construct three header tables with the same ordering of frequent items (see Table 6). Note that we need to keep the items in each header table exactly the same, even though some groups may not include all of these items. The reason is that we need to process three trees simultaneously, and we need to keep the same items in the three header tables.

Table 5 Conditional Pattern Base for d in All Three Trees

Group	Item	Conditional pattern base	Conditional tree
G_1	d	$\{b, c: 1/2\}, \{a: 1/2\}$	$\langle b, c: 1/2\rangle, \langle a: 1/2\rangle$
G_2	d	$\{b, c: 1/3\}, \{b: 1/2\}, \{a: 1/3\}$	$\langle b: 5/6, c: 1/3\rangle, \langle a: 1/3\rangle$
G_3	d	$\{b, c: 1/2\}, \{a: 1/3\}$	$\langle b: 1/2, c: 1/2\rangle, \langle a: 1/3\rangle$

Table 6 Frequent Items Derived from d's Conditional Pattern Base

Item	Support count	Item	Support count	Item	Support count
b	0.50	b	0.83	b	0.50
c	0.50	c	0.33	c	0.50
a	0.50	a	0.33	a	0.33

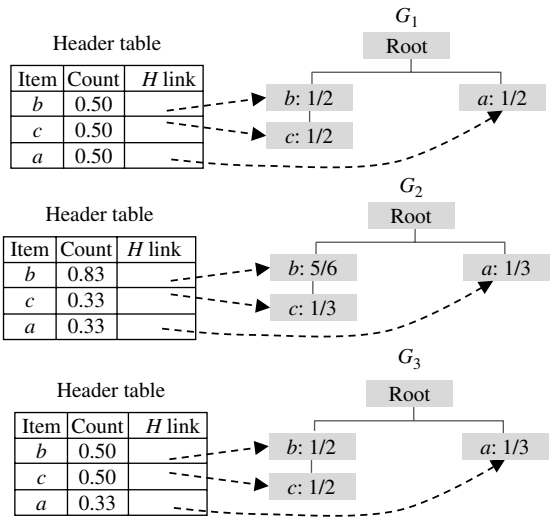


Figure 3 **Conditional Trees for *d***

Based on the conditional pattern base in Table 5 and the frequent items in Table 6, we can create three conditional trees and their corresponding header tables (see Figure 3). The procedure of mining the conditional trees is similar to that of mining the original trees. This is a recursive process (i.e., these conditional trees will generate other conditional trees for the items in their header tables). The entire process will stop when there is no item left (i.e., we have considered all item sets).

4.4. The Overall Process and Formal Algorithm

Figure 4 outlines the formal algorithm for the entire method. Please also see the online supplement for the flow of the entire process in discovering group differences.

Input: A transaction database DB with n groups $\{G_1, G_2, \ldots, G_n\}$, a minimum support min_sup, and a minimum difference min_dif
Output: The complete set of group differences
Major steps:
1. DIFF( ):
2. Create the header table based on the frequent items.
3. Construct a tree for each group.
4. For each item i in the header table (order: bottom up):
5. If i satisfies the pruning condition:
6. Prune branches containing i
7. Go to step 4
8. If i satisfies condition (M4), (M2), and (M3):
9. Output i as a group difference
10. If i has conditional pattern base:
11. Call DIFF( )
12. Else:
13. Go to step 4

Figure 4 **Group Difference Detection Algorithm—DIFF**

In Figure 4, DIFF( ) is a recursive function. Every time we need to process the original transaction database or a conditional pattern base, we call on this function. Given a transaction database or a conditional pattern base, we first collect all frequent items in the database, rank these items, and create the header tables and trees. The header tables for different trees share the same order of items, because we need to synchronize the process of traversing all the trees. The header tables have different count values and node links. When we process the trees, we go through the items in the header table from the bottom up. For each item in the header table, we first check whether all the branches (in all the trees) containing this item can be pruned. If the header table is built based on the conditional pattern base, then each item in the header table corresponds to an item set. For example, when we are processing item b from the header table built based on d's conditional pattern base, we are actually processing the item set $\{d, b\}$. If the pruning condition is satisfied, we prune all related branches from all trees, and move to the next item in the header table. If the pruning condition is not satisfied, we then check whether the group difference satisfies conditions (M2)–(M4). If all three conditions are satisfied, we output this item (or item set if processing the conditional pattern base) as a group difference. As long as this item is not pruned (regardless of whether the current item is a group difference), we find its conditional pattern base and recursively call the function DIFF( ). This recursive process will end after we have processed all possible item sets.

5. Experimental Evaluations

In this section, we compare our method of mining group differences (DIFF) with the STUCCO method proposed in Bay and Pazzani (1999, 2001). We compare the performance of these two methods on completeness of the results, pruning efficiency, and computational efficiency. Section 5.1 reports results on an Internet visitation data set. Section 5.2 reports results on three standard data sets. We show that our method performs better than STUCCO in terms of pruning efficiency and computational efficiency. Because STUCCO does not consider relative difference, to ensure fair comparisons we limit our comparisons between DIFF and STUCCO to absolute difference for most parts of the experiments, and in §5.2 we add in the statistical significance test and relative difference test.

5.1. Internet Visitation Data

In the experiments, we apply both our method and STUCCO to a real data set capturing Web users' online activities (provided by comScore Networks).

We analyze the group differences between different generation Web users.

The data set we use captures the user-centric online behavior of a random sample of 50,000 users over one year. It captures the entire history of Web-surfing behavior for each user. Users are identified by machine IDs, because individual machines are tracked through installed software instead of actual users. Web-browsing activities are recorded in the format of Web sessions. A Web session is a commonly used unit of analyzing Web user behavior. It contains a list of consecutive URLs visited. Typically, industry heuristics draw session boundaries if the time difference between consecutive clicks exceeds a chosen threshold, say 30 minutes. Each session contains a list of websites (domain names) visited within the session.

Each panelist in the sample represents one household that has been tracked. Because we divide users according to age, it is important for the clickstream data to represent a single user's data, as opposed to a household. Hence we restrict our selection of users to those with a household size of one. We acknowledge that this treatment may bias the representativeness of our sample. However, given the large number of single-user households remaining in the data set, we argue that our analysis can still elicit the common features of different generations of users.

Based on the age of the user, we group all single-user households (which we henceforth refer to as users) into three groups, young (Y), middle-aged (M), and senior (S). Table 7 briefly describes each group.

Under the same support threshold (min_sup) and group difference threshold (min_dif), both DIFF and STUCCO find exactly the same set of group differences, which indicates that DIFF discovers all of the qualified group differences.

In real practice, the selection of min_dif, min_sup, and also min_ratio, is flexible. Managers are often interested in the most significant group differences, which correspond to the ones with highest absolute difference and relative difference (lowest ratio). The min_sup value needs to be set low enough to allow us to explore more selections. Given a low min_sup value, we can start with a relatively high min_dif and low min_ratio. If not enough good-quality group differences are discovered under high value of min_dif and low min_ratio, we can adjust these two thresholds to allow more group differences to be discovered. If computation is not an issue, we can set all three thresholds low, and rank list the results according to either min_dif or min_ratio. The relationship between min_dif and min_ratio can also be adjusted. When min_dif is set to be relatively high, we can increase min_ratio to make sure that the item sets with high support and high absolute difference are discovered (because the relative difference will normally decrease as support increases). When min_ratio is low (stricter requirement on relative difference), we still need to set reasonable min_dif value to make sure that item sets with extremely low support can be filtered out. Similar to most problems involving such thresholds, it is hard to pinpoint the exact values for the parameters. However, in the process of experimenting, we can find the right number of good-quality group differences.

Table 7 Definition and Descriptive Statistics for Three Age Groups

Group label	Age	No. of users	No. of sessions
Y (Young)	18–29	1,375	599,860
M (Middle-aged)	30–49	3,102	1,445,702
S (Senior)	50 and older	1,772	754,768

Now, we compare the pruning efficiency of DIFF and STUCCO. According to Property 2, theoretically, our pruning strategy is superior to that of STUCCO. Here, we use experiments to further confirm that DIFF indeed performs much better in terms of pruning efficiency. In Figure 5, under every level of min_dif, the number of nodes pruned in DIFF is higher than that of STUCCO. Also in these data, all of the pruned nodes in STUCCO are contained in the nodes pruned in DIFF. This demonstrates that DIFF has better pruning efficiency compared to STUCCO.

Note, the search space for DIFF is smaller than that for STUCCO because of the compressed tree structure. If the absolute number of pruned nodes is greater for DIFF, it means that the relative pruned search space is even bigger. As we can see from Figure 5, the number of pruned nodes first increases and then decreases with min_dif (areas within the dotted rectangles show the inflection point). As min_dif increases, the number of pruned nodes on the upper levels of the trees also increases. This drastically shrinks the search space, and the nodes on the lower level of the tree will not be traversed at all, which leads to the decrease in the total number of nodes pruned.

In addition, we compare DIFF and STUCCO on execution time. Under three different min_sup levels (5%, 5.5%, 6%), DIFF achieves execution time around 5 seconds, whereas STUCCO's execution time is

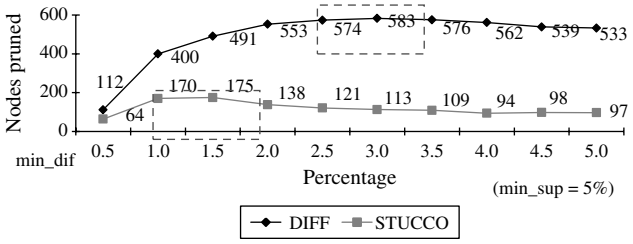


Figure 5 Number of Nodes Pruned Under DIFF and STUCCO as min_dif Varies

around 30 seconds, about six times longer. This clearly illustrates that DIFF is much more efficient than STUCCO.

To summarize, our method DIFF performs much better than STUCCO in terms of pruning efficiency and computational efficiency and also returns the complete set of group differences.

Please also refer to the online supplement for more results on this Internet visitation data.

5.2. Standard Data Sets

To show the robustness of our results, we also compare DIFF and STUCCO using three standard data sets from the UCI Machine Learning Repository (Frank and Asuncion 2010).

We use the adult data set to learn the differences between people whose yearly income is less than $50,000 and people who earn more than $50,000 each year based on census data. This data set was also used by Bay and Pazzani (1999, 2001). The attributes for each user include age, occupation, education, sex, hours worked, salary, etc. We use the mushroom data set to learn the differences between edible and poisonous mushrooms. This data set was also adopted by Bay and Pazzani (2001). Both the adult data set and mushroom data set have two groups. The poker hand data set we use has 10 different groups corresponding to different poker hands. Each card is described using two attributes (suit and rank), and the order of the cards matters as well (see more explanation in Frank and Asuncion 2010).

Next, we compare DIFF and STUCCO on pruning efficiency and computation efficiency. First, we only use the absolute difference, because that is what STUCCO considers. As shown in Table 8, the two methods uncover the same number of group differences, which indicates the completeness of the results. The number of item sets considered refers to the number of item sets for which the methods calculate the support. A lower number means that the pruning efficiency is high. As we can see, DIFF is significantly more efficient than STUCCO. DIFF also has significantly lower execution time. In addition, we compare the results after applying a statistically significance test and relative difference test to the group differences discovered by DIFF, and apply statistical significance test to the group differences discovered by STUCCO. Table 9 lists the results of the number of group differences discovered (the poker data set was not used because the number of data points for some groups is too low to run the chi-squared test for the statistical significance test). DIFF in Table 9 incorporated all three conditions (M2)–(M4), STUCCO incorporated (M3)–(M4), and we also compare with the benchmark case where only (M4) is used.

For Table 9, we set min-ratio to be 0.5. Because we used the relative difference in addition to absolute difference and statistical difference for DIFF, the number of group differences discovered by DIFF can be smaller than that discovered by STUCCO, which only uses absolute difference and statistical significance. If we set min-ratio to be 1 (i.e., ignoring the

Table 8 Comparisons Between DIFF and STUCCO ((M4) Only)

	Adult			Mushroom			Poker		
	min-dif (%)	DIFF	STUCCO	min-dif (%)	DIFF	STUCCO	min-dif (%)	DIFF	STUCCO
No. of group differences	40	27	27	75	14	14	50	7	7
No. of item sets considered		516	7,962		267	4,399		250	465
Time(ms)		1,360	23,343		430	1,532		891	995
No. of group differences	10	2,167	2,167	50	902	902	10	16,031	16,031
No. of item sets considered		8,628	182,275		2,436	50,899		95,928	152,796
Time(ms)		1,937	162,944		709	9,813		2,734	28,786
No. of group differences	1	155,494	155,494	25	46,956	46,956	5	17,295	17,295
No. of item sets considered		315,991	6,774,099		65,566	919,059		107,307	175,970
Time(ms)		4,332	1,231,771		1,748	90,226		2,751	37,229

Note. The min-dif threshold values chosen are different for different data sets, because of the different number of group differences generated.

Table 9 Comparisons Between DIFF and STUCCO (M2)–(M4)

	Adult				Mushroom			
	min-dif (%)	DIFF (M4) + (M2) + (M3)	STUCCO (M4) + (M3)	DIFF/STUCCO (M4)	min-dif (%)	DIFF (M4) + (M2) + (M3)	STUCCO (M4) + (M3)	DIFF/STUCCO (M4)
No. of group differences	40	27	27	27	75	14	14	14
No. of group differences	10	1,991	2,167	2,167	50	902	902	902
No. of group differences	1	144,232	155,049	155,494	25	46,892	46,956	46,956

relative difference condition), DIFF and STUCCO discover exactly the same number of group differences. The DIFF/STUCCO (M4) column in Table 9 provides the number of group differences discovered using just (M4). As we can see, when the min_dif is low (1%), some discovered group differences using just (M4) are not statistically significant, and some of those statistically significant ones do not satisfy the relative difference (i.e., $155{,}494 > 155{,}049 > 144{,}232$). Please also see the online supplement for more results.

6. Conclusions

In this paper, we develop a novel group difference detection method that significantly outperforms the state of the art. The main contribution of the paper is that we provide a more general definition of group difference and provide a significantly better method for group difference detection, which is one of the fundamental tasks in data analysis. Our method not only maintains completeness of results, but also achieves higher computational efficiency through better pruning and faster search. In addition, we provide a more general support definition. Our definition allows managers to choose whether transaction generators should be incorporated. We conduct comprehensive experiments to compare our method with the most popular alternative for group difference detection, STUCCO (Bay and Pazzani 1999, 2001), on completeness of results, pruning efficiency, and computational efficiency. The experiments demonstrate that our method guarantees completeness of results, and achieves higher pruning efficiency and computational efficiency compared to STUCCO.

Our method has limitations: DIFF cannot directly handle continuous attributes. They need to be discretized first and then treated as categorical attributes. In addition, our method is designed to discover the type of group difference defined in this paper. A new method may need to be designed for a different pattern format. For example, our method can only handle conjunction of item sets (X and Y) instead of disjunction of item sets (X or Y). However, we believe our definition incorporating both absolute and relative difference is fairly general and covers many common scenarios.

From a methodological perspective, our work offers several interesting extensions in the future. For instance, we can study the evolution of group differences over time. Incorporating an additional data stream into the group difference discovery process can also facilitate the study of the evolution of group differences. We can also study the problem of finding the top-k item sets with the highest group difference. A naïve way of using our existing method for this problem is to start from a very high min_dif (e.g., 100%) or very low min-ratio (e.g., 0%), and slowly lower down min_dif or increase min-ratio until we find k group differences. If k is relatively small, this approach can be fairly efficient. If k is big, calling DIFF repeatedly can be time consuming. In the future, we can design a new method to optimally solve this top-k problem.

We show that a single decision tree is not able to output the complete set of significant group differences (see the online supplement). There are also ensemble classifiers, such as Random Forests (Breiman 2001), which outputs the class that is the mode of the classes output by individual trees. However, the process of Random Forests does not guarantee the completeness of the results either, especially when the dimension of the data is high. Even though it is not hard to draw this conclusion, it may still be worthwhile to do a comprehensive study in the future to see whether there are ways to design an ensemble method based on decision trees to generate the complete set of significant group differences efficiently.

On the application side, the method developed in this paper can be adapted to various real-world problems. Aside from applications analyzing differences in Web users, we can also examine how different groups of offline shoppers behave differently in terms of their shopping behavior. This method can be easily applied to investigate promotion effectiveness by comparing groups receiving promotions with groups not receiving promotions. Our paper addresses a more comprehensive type of group differences than those studied by traditional data analysis tools. Therefore, we hope that adopting this method in a wide range of applications can uncover new insights.

Supplemental Material

Supplemental material to this paper is available at http://dx.doi.org/10.1287/ijoc.2013.0558.

Acknowledgments

This work was supported by the National Natural Science Foundation of China [Grants 71272029 and 71110107027].

References

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. *Proc. 20th VLDB Conf. Santiago, Chile*, 487–499.

Alqadah F, Bhatnagar R (2009) Discovering substantial distinctions among incremental bi-clusters. *Proc. 9th SIAM Internat. Conf. Data Mining, Nevada*, 197–208.

Bay SD, Pazzani MJ (1999) Detecting change in categorical data: Mining contrast sets. *ACM SIGKDD Internat. Conf. Knowledge Discovery and Data Mining, San Diego*, 302–306.

Bay SD, Pazzani MJ (2001) Detecting group differences: Mining contrast sets. *Data Mining Knowledge Discovery* 5(3):213–246.

Breiman L (2001) Random Forests. *Machine Learn.* 45(1):5–32.

Breiman L, Friedman J, Stone CJ, Olshen RA (1984) *Classification and Regression Trees* (Chapman and Hall, New York).

Darity WA (2000) Racial and ethnic economic inequality: The international record. *Amer. Econom. Rev.* 90(2):308–311.

Deng K, Zaïane OR (2009) Contrasting sequence groups by emerging sequences. *Lecture Notes in Computer Science*, Vol. 5808/2009 (Springer, Berlin), 377–384.

Dong G, Li J (1999) Efficient mining of emerging patterns: Discovering trends and differences. *Proc. Fifth ACM SIGKDD Internat. Conf. Knowledge Discovery Data Mining* (ACM, New York), 43–52.

Fan H, Ramamohanara K (2003) A Bayesian approach to use emerging patterns for classification. *Proc. 14th Australasian Database Conf. (ADC-03), Adelaide, Australia*, 39–48.

Fan H, Fan M, Ramamohanarao K, Liu M (2006) Further improving emerging pattern based classifiers via bagging. *Proc. 10th Pacific-Asia Conf. Knowledge Discovery Data Mining (PAKDD-06), Singapore*, 91–96.

Frank A, Asuncion A (2010) UCI Machine Learning Repository. School of Information and Computer Science, University of California, Irvine. Accessed August 2011, http://archive.ics.uci.edu/ml.

Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining Knowledge Discovery* 8(1):53–87.

Hilderman RJ, Peckham T (2007) Statistical methodologies for mining potentially interesting contrast sets. *Stud. Comput. Intelligence* 43:153–177.

Kobyliński L, Walczak K (2011) Efficient mining of jumping emerging patterns with occurrence counts for classification. *Transactions on Rough Sets XIII*. Lecture Notes in Computer Science, Vol. 6499/2011 (Springer, Berlin), 73–88.

Kralj P, Lavrac N, Gamberger D, Krstačić A (2007a) Contrast set mining for distinguishing between similar diseases. *Proc. 11th Conf. Artificial Intelligence in Medicine (AIME-07)* (Springer, Berlin), 109–118.

Kralj P, Lavrac N, Gamberger D, Krstacic A (2007b) Contrast set mining through subgroup discovery applied to brain ischaemia data. *Proc. 11th Pacific-Asia Conf. Adv. Knowledge Discovery and Data Mining: (PAKDD-07)* (Springer, Berlin), 579– 586.

Lavrac N, Kavsek B, Flach PA, Todorovski L (2004) Subgroup discovery with CN2-SD. *J. Machine Learn. Res.* 5:153–188.

Liu B, Hsu W, Ma Y (2001) Discovering the set of fundamental rule changes. *Proc. 7th ACM SIGKDD Internat. Conf. Knowledge Discovery Data Mining (KDD-01)* (ACM, New York), 335–340.

Liu B, Hsu W, Han H-S, Xia Y (2000) Mining changes for real-life applications. *Proc. 2nd Internat. Conf. Data Warehousing Knowledge Discovery (DaWaK-2000)* (Springer, Berlin), 337–346.

Loekito E, Bailey J (2006) Fast mining of high dimensional expressive contrast patterns using zero-suppressed binary decision diagrams. *Proc. 12th ACM SIGKDD Internat. Conf. Knowledge Discovery Data Mining* (ACM, New York), 307–316.

Loekito E, Bailey J (2008) Mining influential attributes that capture class and group contrast behavior. *Proc. 17th ACM Conf. Inform. Knowledge Management, Napa Valley, CA,* 971–980.

Minaei-Bidgoli B, Tan PN, Punch WF (2004) Mining interesting contrast rules for a web-based educational system. *Internat. Conf. Machine Learn. Appl.* (IEEE Computer Society, Louisville, KY), 1–8.

Nazeri Z, Barbara D, De Jong K, Donohue G, Sherry L (2008) *Contrast-Set Mining of Aircraft Accidents and Incidents* (Springer-Verlag, Berlin, Heidelberg).

Quinlan JR (1993) *C4.5: Programs for Machine Learning* (Morgan Kaufman Publishers, San Francisco).

Ramamohanarao K (2010) Contrast pattern mining and its application for building robust classifiers. Pfahringer B, Holmes G, Hoffmann A, eds. *Discovery Science*. Lecture Notes in Computer Science, Vol. 6332/2010 (Springer, Berlin), 380.

Ramamohanarao K, Bailey J, Fan H (2005) Efficient mining of contrast patterns and their applications to classification. *Proc. Third Internat. Conf. Intelligent Sensing Inform. Processing* (IEEE Computer Society, Washington, DC), 39–47.

Ruggles S (1997) The rise of divorce and separation in the United States, 1880–1990. *Demography* 34(4):455–466.

Satsangi A, Zaiane OR (2007) Contrasting the contrast sets: An alternative approach. *11th Internat. Database Engrg. Appl. Sympos., Alberta, Canada*, 114–119.

Siu KKW, Butler SM, Beveridge T, Gillam JE, Hall CJ, Kaye AH, Lewis RA, et al. (2005) Identifying markers of pathology in SAXS data of malignant tissues of the brain. *Nuclear Instruments Methods Phys. Res. Sect. A* 548(1–2):140–146.

Song HS, Kim JK, Kim SH (2001) Mining the change of customer behavior in an internet shopping mall. *Expert Systems Appl.* 21(3):157–168.

Wang K, Zhou S, Fu AW-C, Yu JX (2003) Mining changes of classification by correspondence tracing. *Proc. 3rd SIAM Internat. Conf. Data Mining (SDM-03), San Francisco,* 95–106.

Webb GI (2000) Efficient search for association rules. *Sixth ACM SIGKDD Int. Conf. Knowledge Discovery Data Mining, New York*, 99–107.

Webb GI (2001) Magnum Opus version 1.3. Computer software, Distributed by Rulequest Research, http://www.rulequest.com.

Webb GI, Butler SM, Newlands D (2003) On detecting differences between groups. *Proc. ACM SIGKDD Internat. Conf. Knowledge Discovery Data Mining* (ACM, New York), 256–265.

Wong T, Tseng K-L (2005) Mining negative contrast sets from data with discrete attributes. *Expert Systems Appl.* 29(2):401–407.