Boosting Item Findability: Bridging the Semantic Gap between Search Phrases and Item Information

Authors

Department of Computer Science
X University,
Address
Email

Abstract: Most search engines do their text query and retrieval based on keyword phrases. However, publishers cannot anticipate all possible ways in which users search for the items in their documents. In fact, many times, there may be no *direct keyword match* between a search phrase and descriptions of items that are perfect "hits" for the search. We present a highly automated solution to the problem of *bridging the semantic gap* between item information and search phrases. Our system can learn rule-based definitions that can be ascribed to *search phrases* with dynamic connotations by extracting structured item information from product catalogs and by utilizing a frequent itemset mining algorithm. We present experimental results for a realistic e-commerce domain. Also, we compare our rule-mining approach to vector-based relevance feedback retrieval techniques and show that our system yields definitions that are easier to validate and perform better.

Keywords: E-commerce, Data Mining, Frequent Itemsets, Web Data, Information Retrieval, Information Extraction, Relevance Feedback.

1. Introduction

Most search engines do their text query and retrieval using keywords. The average keyword query length is under three words (2.2 words [5]). Recent research [3] found that 40 percent of companies rate their search tools as "not very useful" or "only somewhat useful." Further, a review of 89 sites [3] found that 75 percent have keyword search engines that fail to retrieve important information and put results in order of relevance; 92 percent fail to provide guided search interfaces to help offset keyword deficiencies [3], and seven out of 10 web shoppers were unable to find products using the search engine, even when the items were stocked and available.

The Defining Problem: Publishers cannot anticipate all possible ways in which users search for the items in their documents. In fact, many times, there may be no *direct keyword match* between a search phrase and descriptions of items that are perfect "hits" for the search. For example, if a shopper uses "motorcycle jacket" then, unless the publisher or search engine knows that every "leather jacket" is a "motorcycle jacket", it cannot produce all matches for user's search. Thus, for certain phrases, there is a ***semantic gap*** between the search phrase used and the way the corresponding matching items are described. A serious consequence of this gap is that it results in unsatisfied customers. Thus *there is a critical need to boost item findability by bridging the semantic gap that exists between search phrases and item information*. Closing this gap has the strong potential to translate web search traffic into higher conversion rates and more satisfied customers.

Issues in Bridging the Semantic Gap: We denote a search phrase to be a "*target search phrase*" if does not directly match certain relevant item descriptions. The semantics of items matching such "*target search phrases*" is *implicit* in their descriptions. For phrases with fixed meanings i.e. their connotations do not change such as in "animal print comforter", it is possible to close the gap by extracting their meaning with a thesaurus [21] and relating it to product descriptions, such as "zebra print comforter" or "leopard print bedding" etc. Where they pose a more interesting challenge is when their meaning is subjective, driven by perceptions, and hence their connotations change over time as in the case of "fashionable handbag" and "luxury bedding". The concept of a fashionable handbag is based on trends, which change over time, and correspondingly the attribute values characterizing such a bag also changes. Similarly, the concept of "luxury bedding" depends on the brands and designs available on the market that are considered as luxury and their attributes. Bridging the semantic gap therefore is in essence the problem of inferring the meaning of search phrases in all its nuances.

Our Approach: In this paper we present an algorithm that (i) structures item information and (ii) uses a frequent itemset mining algorithm to learn the "target phrase" definitions**.**

Our Contributions: We present a novel extraction and mining approach for inferring the meaning of search phrases from keyword matching product information. The mined rules can be used by a publisher or a search engine to boost the item findability. We present algorithms for:

- Automated techniques for extracting attribute-value pairs from descriptive item names;
- An optimized heuristic synthesis, on so called 2-frequent itemset graph, for frequent itemset mining.

Next section discusses related work. In Section 3, the architecture of the system is presented. In Section 4, we present the item name structuring algorithm. In Section 5, we present the phrase definition mining algorithm for structured data. In Section 6, we present the experimental results and comparison with relevance feedback [22] method and Section 7 concludes the paper.

2. Related Work

In [14] linguistic analysis is employed to mine the descriptions of phrases/queries. Specifically, the work is based on patterns such as *is a, or, such as, especially, including, or other, and other, etc.,* in order to recognize the meaning of a phrase/query. This approach, however cannot work well with domains where the target phrase does not associate within the context of the above patterns. In [2], *generalized episodes* and *episode rules* are used for Descriptive Phrase Extraction. *Episode rules* are the modification of association rules and *episode* is the modification of frequent set. An *episode* is a collection of feature vectors with a partial order; authors claimed that their approach is useful in phrase mining in Finnish, a language that has the relaxed order of words in a sentence. In [15], authors present a co-occurrence clustering algorithm that identifies phrases that frequently co-occurs with the target phrase from the meta-tags of Web documents. However, in this paper we address a different problem; we attempt to mine the phrase definitions in terms of extracted item information, thus, the mined definitions can be utilized to connect "search phrases" to real items in all their nuances.

The frequent itemset mining problem is to discover a set of items shared among a large number of records in the database. There are two main search strategies to find the frequent items set. Apriori [1] and several other Apriori like algorithms adopt Breadth-First-Search model, while Eclat [17] and FPGrowth [8] are well known algorithms that employ Depth-First manner to search all frequent itemsets of a database. Our algorithm also searches for frequent itemsets in a Depth-First manner. But, unlike the lattice structure used in Eclat or the conditional frequent pattern tree used in FPGrowth, we propose the so called 2-frequent itemset graph and utilize heuristic syntheses to prune the search space in order to improve the performance. We plan to further optimize our algorithm and conduct detailed comparisons to the above algorithms.

The relevance feedback [22] method can also be used to refine the original keyword phrase by using the document vectors [23] of the extracted relevant items as additional information. In Section 6, we present experimental results and show that the rules that our system learns, by utilizing the extracted relevant item information, are easier to validate and perform better than retrieval with the relevance feedback method.

3. System Description

I. Item Name Structuring: This component takes a product catalogue and extracts structured information for mining the *phrase based* and *parametric* definitions. Details are discussed in Section 4.

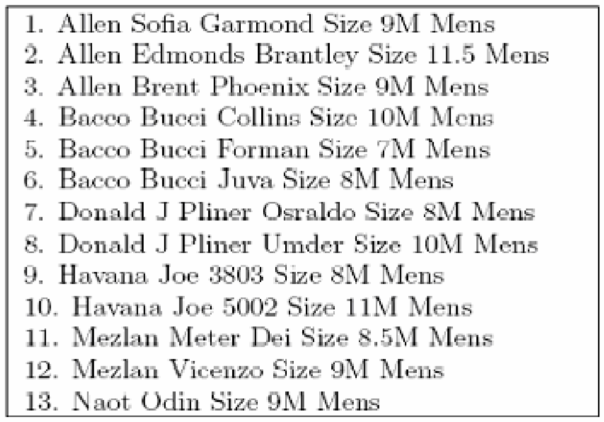
II. Mining Search Phrase Definitions: In this phase, we divide the phrase definition mining problems into two sub problems (i) mining the parametric definitions from extracted attribute value pairs of items, and (ii) mining phrase based definitions from the long item descriptions. Details are discussed in Section 5.

4. Data Labeling

This section presents the techniques for an e-commerce domain, for the sake of providing examples. Our techniques can be customized for different domains. The major tasks in this phase are *structuring and labeling* of extracted data.

4.1 Labeling and Structuring Extracted Data

This section describes a technique to partition the short product item names into their various attributes. We achieve this by grouping and aligning the tokens in the item names such that the instances of the same attribute from multiple products fall under the same category indicating that they are of similar types. The input to our algorithm is unlabelled, short, attribute rich product names of similar items collected from a web page. An example collection of shoe names that this algorithm processes is shown in Figure 1. The desired output for this data set is the partitioned and aligned unlabelled product attributes shown in Table 2. Our algorithm is completely unsupervised and does not need any labeled training data. It uses frequent word patterns and context information to identify and group attributes present in the product names.
The motivation behind doing the partition is to organize data. By discovering attributes in product data and arranging the values in a table, one can build a search engine which can enable quicker and precise product searches in an efficient way. One may think why extracting attribute information from product names is indeed a difficult problem.



1. Allen Sofia Garmond Size 9M Mens
2. Allen Edmonds Brantley Size 11.5 Mens
3. Allen Brent Phoenix Size 9M Mens
4. Bacco Bucci Collins Size 10M Mens
5. Bacco Bucci Forman Size 7M Mens
6. Bacco Bucci Juva Size 8M Mens
7. Donald J Pliner Osraldo Size 8M Mens
8. Donald J Pliner Umder Size 10M Mens
9. Havana Joe 3803 Size 8M Mens
10. Havana Joe 5002 Size 11M Mens
11. Mezlan Meter Dei Size 8.5M Mens
12. Mezlan Vicenzo Size 9M Mens
13. Naot Odin Size 9M Mens

Figure 1: Example of a Collection of Shoe Names

The aspect of this problem that makes it challenging can be attributed to two properties of such data. First, the products that are presented in list-of-product pages of online stores may really not have similar attribute information after all! The taxonomy the store uses to organize its data may not be well set up so that products at a particular leaf node can have similar attributes. Secondly, even if similar products are grouped in a single page, the way their names are presented may still be different. Multiple tokens for a single type, missing attributes in the descriptions, interchanged order, ambiguity in tokenization are some important barriers to formulating an algorithm that consistently performs well in all situations.

4.2 The Algorithm

Since the item names are all obtained from a single source and belong to the same category, they are likely to have a similar pattern. As mentioned before, our algorithm is designed to process collections of such item names without any labeling whatsoever. So it can be performed on the fly as and when data is extracted from the web sites. Following are the general properties of the data our algorithm can process:

- **Super-Tokens**: Any pair of tokens t_1, t_2 that always co-occur together and occur more than once belong to a multi token instance of a type.
- **Context**: All single tokens occurring between identical attribute types belong to the same type. This means that if two tokens t_1 and t_2 from distinct item names occur in between same types T_L and T_R then they should be of the same type.
- **Anchor Type**: A token that uniquely occurs within all item names should belong to a unique type, which we call an *Anchor Type*.
- **Density**: Attribute types should be densely populated. Meaning that, every type should occur within the majority of item names.
- **Ordering:** Pairwise ordering of all types should be consistent within a collection.

The more the input data adheres to the above mentioned properties, the better the results produced will be. Tolerance to deviation in these characteristics is very reasonable.

The high level steps of the *Item Name Partition* algorithm are: Tokenization, Super Tokenization, Context Based Inference, Type Ordering, Density based Merging and Concatenation. The execution steps are shown in Algorithm 1.

Tokenization: The item names are tokenized by using white space characters as delimiters. Tokens are stemmed so using the Porter Stemmer [11]. Each item name D_i is converted into a sequence of tokens t_1, t_2...

Super Tokenization: The second step identifies multi-token attributes like `Donald J Pliner', `Bacco Bucci' and `Havana Joe' for the example in Figure 1. This is done by concatenating any sequence of tokens that are always neighboring to each other and co-occurring more than once, into a single super-token.

Algorithm 1: Item Name Partition

```
Input: D, a collection of item names $D_0$, $D_1$,...$D_n$
Output: T, an ordered set of Types $T_0$, $T_1$,...$T_m$ where $T_i$ is a bag of
tokens that are instances of the same product attributes

1: Tokenization(D)
2: SuperTokenization(D)
3: for each unique token $t_i \varepsilon D_i$ where $D_i \varepsilon$ D do
4:    create new type $T_i$
5:    add $t_i$ to $T_i$
6:    add $T_i$ to T
7: end for
8: repeat
9:    ContextBasedInference($D$, $T$)
10: until no inference is made
11: SortTypes($D$, $T$)
12: repeat
13:    MergeTypes($D$, $T$)
14: until no types are merged
15: repeat
16:    MergeConcatenateTypes($D$, $T$)
17: until no types were merge-concatenated
```

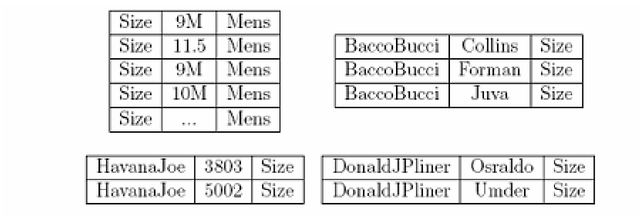
Initialization of Types: To initialize, every item name is prefixed and suffixed with a *Begin* and an *End* token. Next, for every unique token t_i in the collection of item names **D**, a new type T_i is created with $t_i \in T_i$.

Context Based Inference: This step aligns tokens from different item names under a single type. This step takes advantage of tokens repeating across descriptions and operates based on the first assumption, Context, that tokens within similar contexts have similar attribute types.

If a token sequences t_x,t, t_y and t'_x, t', t'_y exist in D such that t_x, $t'_x \in T_p$ and t_y, $t'_y \in T_q$, then combine and replace the types of tokens t and t' with a new type T_n = Typeof(t) U Typeof(t') .

As an example, the tokens `Collins' and `Forman' occur between the same pair of tokens `Bacco Bucci' and `Size'. So the Types of these two tokens are combined to form a new type containing both tokens. Context based inference is applied successively applications until it can not be applied anymore. The applications of Context Based Inference for the example in Figure 1 are shown in Figure 2.

Type Ordering: In this step, the set of inferred types T are sorted based on their ordering in the original item names. We utilize the Pairwise Offset Difference (POD) metric to compare different types.



Size	9M	Mens
Size	11.5	Mens
Size	9M	Mens
Size	10M	Mens
Size	...	Mens

BaccoBucci	Collins	Size
BaccoBucci	Forman	Size
BaccoBucci	Juva	Size

HavanaJoe	3803	Size
HavanaJoe	5002	Size

DonaldJPliner	Osraldo	Size
DonaldJPliner	Umder	Size

Figure 2: Application of Context Based Inference for examples in Figure 1

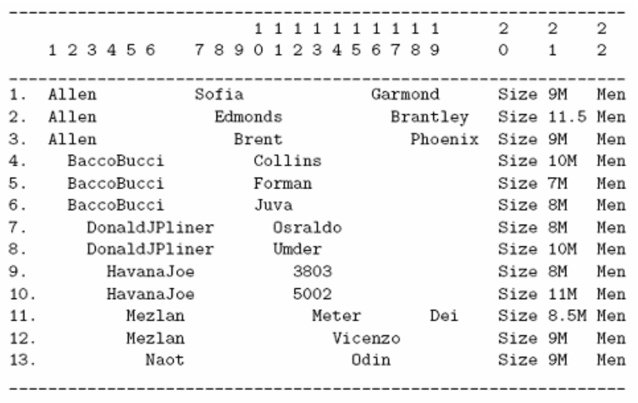
POD between types T_i and T_j is defined as:

$$POD_{ij} = \sum_{x \epsilon T_i, y \epsilon T_j} (f_x - f_y) \qquad (1)$$

where f_x is the token offset of x from the start of its item name and f_y is the token offset of y. If this value is greater than zero, then the type T_i comes after type T_j in the sorted order.

The example descriptions up to the ordering step can be visualized as shown in Figure 3. Each column in the table corresponds to a type inferred from the data in Figure 1. Due to space constraints, tokens have been aligned such that those from the same type are offset at the same column. The type numbers the tokens belong to are indicated at the top.

Type Merging: A careful observation of Figure 3 shows that some of the neighboring types are fillers for each other. Meaning that, they are not instantiated together for any item name. Such types are candidates for merging and are called *merge compatible*. Merging at this point is logical because of our assumption that the types are densely populated.



```
---------------------------------------------------------------
                         1 1 1 1 1 1 1 1 1 1      2    2    2
    1 2 3 4 5 6    7 8 9 0 1 2 3 4 5 6 7 8 9      0    1    2
---------------------------------------------------------------
1.  Allen          Sofia             Garmond      Size 9M   Men
2.  Allen            Edmonds           Brantley   Size 11.5 Men
3.  Allen              Brent             Phoenix  Size 9M   Men
4.    BaccoBucci         Collins                  Size 10M  Men
5.    BaccoBucci         Forman                   Size 7M   Men
6.    BaccoBucci         Juva                     Size 8M   Men
7.      DonaldJPliner      Osraldo                Size 8M   Men
8.      DonaldJPliner      Umder                  Size 10M  Men
9.        HavanaJoe          3803                 Size 8M   Men
10.       HavanaJoe          5002                 Size 11M  Men
11.         Mezlan             Meter       Dei    Size 8.5M Men
12.         Mezlan               Vicenzo          Size 9M   Men
13.           Naot                 Odin           Size 9M   Men
---------------------------------------------------------------
```

Figure 3: The Examples in Figure 1 after Type Ordering

The next best pair of types to merge is determined by scoring all merge compatible types and choosing the pair with the highest score. The scoring metric for a pair of merge compatible types is the size of their merged type. To resolve ambiguities, the following variation of the POD distance in Equation (1) between the ambiguous pairs is used and the closest one is chosen. Here the absolute distances are summed up to ignore the ordering between types.

$$POD^a_{ij} = \sum_{x \epsilon T_i, y \epsilon T_j} |f_x - f_y| \qquad (2)$$

To prevent irrelevant and distant types from being merged together, only those merge compatible types that are between consecutive Anchor Types in the ordered types are considered. The columns T_{20} and T_{22} in Figure 3 correspond to Anchor Types. Merging for this example is done for the Types 1 to 6, followed by merging of types 7 to 15, and 16 to 19. The result is shown in tabular form in Table 1.

Merge Concatenation: Finally, merge-concatenation is performed to eliminate sparsely populated types. Sparsely populated types are those with a majority of missing values. By our assumption, collections of item names should have dense attributes. This implies that the tokens of a sparsely populated type should be concatenated with the values of one of the neighboring types. The closest neighboring type within its anchor area is found using the absolute POD metric in Equation (2). For the example in Table 1, Type 16 is classified as a sparse types, and since Type 7 is its closest neighbor, its values are appended to corresponding values of Type 7 yielding the final results/alignment presented in Table 2.

4.3 Experimental Results

To evaluate the algorithm, our DataRover system was used to crawl and extract list-of-products from the following five Web sites.

- www.officemax.com

- www.officedepot.com
- www.acehardware.com
- www.homeclick.com
- www.overstock.com

The first two sites feature office supplies. The third and the fourth sites sell hardware goods for construction and home improvement. The last site features a variety of products from various domains including apparel, electronics, home & garden etc. An average of around 250 list-of-product pages were extracted from each site and their corresponding sets of product names were processed by this algorithm.

Random samples of about 25 sets of product names for the first two and 10 for the remaining 3 sites were collected for output analysis. The type-value pairs and attributes were manually inspected and the evaluation measures were calculated for each site.

Three metrics were used to measure the effectiveness of the algorithm. The first two evaluate the ability to identify fragments of the descriptions to the correct type and the last one indicates the correctness of the number of attributes.

No.	T_1	T_7	T_{16}	T_{20}	T_{21}	T_{22}
1	Allen	Sofia	Garamond	size	9M	Men
2	Allen	Edmonds	Brandley	size	11.5	Men
3	Allen	Brent	Phoenix	size	9M	Men
4	Bacco Bucci	Collins		size	10M	Men
5	Bacco Bucci	Forman		size	7M	Men
6	Bacco Bucci	Juva		size	8M	Men
7	Donald J Pliner	Osraldo		size	8M	Men
8	Donald J Pliner	Umder		size	10M	Men
9	Havana Joe	3803		size	8M	Men
10	Havana Joe	5002		size	11M	Men
11	Mezlan	Meter	Dei	size	8.5M	Men
12	Mezlan	Vicenzo		size	9M	Men
13	Naot	Odin		size	9M	Men

Table 1: Partition Visualized after Type Merging

Precision in type-value pair detection: This quantity indicates how correctly type-value pairs are identified. A type-value pair is said to be identified correctly if every single token belonging to that value of the attribute being described have been identified correctly. It is calculated as

$$Precision = \frac{\text{type-value pairs identified correctly}}{\text{type-value pairs identified}}$$

Recall in type-value pair detection: This quantity indicates if every existing type-value pair is being identified. A lower value for this measure indicates that not all type-value instances are being captured by the algorithm. It is calculated as

$$Recall = \frac{\text{type-value pairs identified correctly}}{\text{type-value pairs existing in the data}}$$

No.	T_1	T_7	T_{20}	T_{21}	T_{22}
1	Allen	Sofia Garamond	size	9M	Men
2	Allen	Edmonds Brandley	size	11.5	Men
3	Allen	Brent Phoenix	size	9M	Men
4	Bacco Bucci	Collins	size	10M	Men
5	Bacco Bucci	Forman	size	7M	Men
6	Bacco Bucci	Juva	size	8M	Men
7	Donald J Pliner	Osraldo	size	8M	Men
8	Donald J Pliner	Umder	size	10M	Men
9	Havana Joe	3803	size	8M	Men
10	Havana Joe	5002	size	11M	Men
11	Mezlan	Meter Dei	size	8.5M	Men
12	Mezlan	Vicenzo	size	9M	Men
13	Naot	Odin	size	9M	Men

Table 2: Partition Visualized at the end of the Algorithm after Type Concatenations

Error Rate in the number of attributes detected: This quantity indicates how well the algorithm detects the number of attributes described in the set of product names. It is calculated as

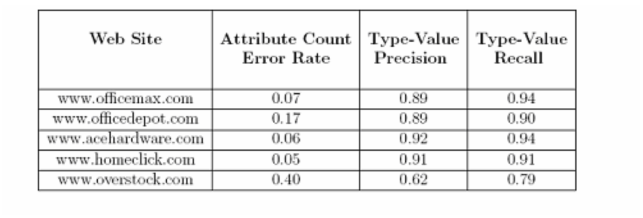
$$Error = \frac{|\text{actual attribute count - guessed attribute count}|}{\text{actual attribute count}}$$

The summary of average measures for each web site is shown in Table 3.

Before trying to analyze the graphs, it is helpful to be aware of the different kind of errors that are possible, and how these errors affect the evaluation measures.

- Firstly, types can be wrongly populated. Tokens from a description can be misplaced in the wrong type. This error will affect the count of type-value pairs correctly identified and hence will lower the precision as well as the recall.
- Secondly, multiple types could be incorrectly merged. This happens when one of the types are sparsely populated. Since the algorithm works based on the fact that types are densely populated, type that are sparsely populated are merge-concatenated with an other neighboring type. This causes a reduction in recall because not all the existing type-value pairs will be identified. Also, the attribute count error will increase because of the missing type.
- Finally, types may remain split since most of the values in the type could have been multi-token values. This situation, where types are split but are densely populated does not force a merge. This will result in extra types in the result and will affect all three measures negatively.

The four Web sites are mainly from two domains of products which are attribute rich - office supplies and hardware. To demonstrate the performance when these assumptions are violated, we also chose a fifth web site, Overstock, that sells a wide range of products not necessarily attribute rich, whose data sets deviate away from the assumptions.



Web Site	Attribute Count Error Rate	Type-Value Precision	Type-Value Recall
www.officemax.com	0.07	0.89	0.94
www.officedepot.com	0.17	0.89	0.90
www.acehardware.com	0.06	0.92	0.94
www.homeclick.com	0.05	0.91	0.91
www.overstock.com	0.40	0.62	0.79

Table 3: Summary of Evaluation Measures for Different Web Sites for the Items Name Structuring Algorithm

5. Mining the Definition of a Target Phrase

In this section, we introduce the problem of mining definitions of a phrase from product data extracted from the matching Web pages. Using extraction techniques discussed in Section 4 we can retrieve tabular parametric attributes of matching products as well as their long descriptions. Next, we apply frequent itemset mining algorithms to learn the parametric definitions and phrase-based definitions of target phrases from the extracted product data.

First, in Sections 5.1 thru 5.4 we introduce an algorithm that finds all frequent itemsets from a database. Section 5.5 discusses the problem of mining parametric definitions. In Section 5.6 textual definition mining is discussed.

Since their introduction in 1994 by Agrawal et al. [1], the frequent itemset and association rule mining problems have received a lot of attention among data mining research community. Over the last decade, many research papers [1, 6, 8, 17, 9] have been published presenting new algorithms as well as improvements on existing algorithms to tackle the efficiency of frequent itemset mining problems. The frequent itemset mining problem is to discover a set of items shared among a large number of transaction instances in the database. For example, consider the product information database matching '*trendy shoes'* that we extract from retail Web sites. Here, each instance represents the collection of product's <attribute, value> pairs for attributes such as brand, price, style, gender, color and description (See Figure 11). The discovered patterns would be the set of <attribute, value> pairs that most frequently co-occur in the database. These patterns define the parametric description of the target phrase '*trendy shoe'*.

The main features of our frequent itemset mining algorithm are as follows:

1. It uses the vertical Boolean representation to encode the database into main memory. Specifically, in the database, each column is encoded as a Boolean array (bit vector) whose length is exactly the number of instances in the data set. Each element of the attribute vector represents the value of that attribute (occurs or not occur) in the corresponding record (instance)
2. We define and construct the 2-frequent itemset Graph and search for all frequent itemsets by utilizing this Graph.
3. We introduce two heuristic functions based on 2-frequent itemset Graph to prune the search space.
4. We also discuss optimization techniques that exploit the internal structure of our itemsets.

5.1 Boolean representation of the database.

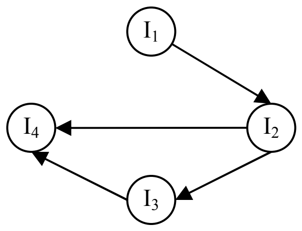
The advantage of Boolean representation is that many logical operations such as superset, subset, set subtraction, OR, XOR, etc between any number of attribute vectors can be performed extremely fast.

5.2 Constructing 2-frequent Itemsets Graph.

The set of 2-frequent itemsets plays crucial role in finding all frequent itemsets. The main idea is that, from the observation that if $\{I_{i....}I_j\}$ is a frequent itemset then all pairs of items in this set must also be a frequent itemset. Using this property of a frequent set, our algorithm will first create a graph that represents the 2-frequent itemsets among all items that satisfy the minimum support threshold.

The the 2-frequent itemset graph is the directed graph G(V,E) which is constructed as follows:
V = *I*; *I* is the set of items that satisfy the minimum support in database *D*.
E = $\{(v_i,v_j) \mid \{i,j\}$ is a 2-frequent itemset and $i<j)$.
We sort the frequent single items into lexicographical order and for a 2-frequent itemset, we construct a directed edge from the node (item) whose index is lower to the node whose index is higher.

Example 1.



I_1	I_2	I_3	I_4
1	1	1	0
0	1	1	1
1	0	1	0
1	1	0	0
0	0	1	1
1	1	0	0
0	1	0	1
1	0	0	1

Figure 4. Database I and its 2-frequent item graph

For this database, if minimum support δ is set to 25%, then the 2-frequent itemsets are I_1I_2, I_2I_3, I_2I_4, I_3I_4. The 2-frequent itemsets graph would be as in Figure 12.

5.3 Searching for Frequent Itemsets

The algorithm iteratively starts from every node in the graph and recursively traverses depth-first to its descendants. At any step k (k>1), the algorithm will choose to go to a child node *v* of the current node so that the path from the beginning node to *v* forms a k-frequent itemset. If so, the algorithm will continue expand to *v*'s children to search for (k+1)-frequent itemset and so on. There are several algorithms [8, 17] that generate frequent itemsets in depth-first manner. A distinguishing feature of our algorithm is that it searches on the 2-frequent itemset graph.

Finding all 2-frequent set takes $O(n^2)$ operations where n is the number of frequent single items. Our algorithm utilizes the following heuristics to guide the search.

Heuristic 1: At step k, choose only children nodes of node v_{k-1} that have incoming degree greater than or equal to the number of visited nodes, counted from the beginning node. Incoming degree of a node *v*, denoted as deg(*v*) is the number of nodes that point to *v*. The meaning of this heuristic is that, if deg(*v*) is smaller than the number of visited nodes (nodes in the path) then there exists at least one node among the set of previously visited k-1 nodes that does not point to *v*. In other words, there exists at least one node in the current path that does not form a 2-frequent itemset with v. Therefore the k-1 nodes in the path (visited nodes) and *v* cannot form a k-frequent itemset hence it is pruned out without candidate itemset generation.

Heuristic 2: At step k, choose only children nodes of node v_{k-1} that have the set of incoming nodes that is a superset of the set of all k-1 nodes in the visited path. This heuristic, which is applied after Heuristic 1, ensures that all previously visited nodes in the current path, must point to the node in consideration. This is also a necessary precondition that each visited node forms a 2-frequent itemset with the node in consideration.

Heuristic 1 is efficient since the 2-frequent itemset graph is already constructed and the degree of all nodes is stored before the search proceeds. Heuristic 2 superset testing operation can also be performed efficiently using the bit-vector representation. Consequently, by utilizing these heuristic estimates, we can prune a lot of nodes that cannot be added to the visited nodes to form a frequent itemset and eliminate a lot of candidate itemset generation.

Algorithm 2: Frequent Itemset Mining

```
Input: D, δ, I
Output: all frequent itemsets F(I,δ)
//main procedure
Construct all 2-frequent itemsets by counting the support of
all pairs of single items
Construct the 2-frequent itemsets graph G(V,E)
F[I, δ]:= {}
for all v in V do
        visited node list:= null
        add v to list of visited node list
        calculate_itemset (v)
endfor
end

Procedure calculate_itemset (v)
for each child v' of v do
    // Heuristic 1
    if deg(v') ≥ number of visited node then
        // Heuristic 2
        if the set of incoming nodes of v' is a superset of
        visited node then
            if the set of visited node and v' has counting ≥ δ
            then
                add v' to the list of visited nodes
                add visited nodes list to F(I, δ)
                calculate_itemset(v')
                //backtrack
                remove v' from the visited node set
            endif
        endif
    endif
endfor
end
```

Example. We consider the database in Figure 12 and its 2-frequent itemset graph.

The search algorithm starts from vertex I_1, I_1 has only one child i.e. I_2. Obviously, I_1 and I_2 form a 2-frequent itemset. The algorithm then goes on to I_2's children. I_2 has two children i.e. I_3 and I_4. Heuristic 1 tells the algorithm that $deg(I_3) = 1$ is smaller than the number of visited nodes (so far, the algorithms has already visited I_1 and I_2 in this iteration). Hence, the algorithm does not consider the set $\{I_1, I_2, I_3\}$. In other words, the algorithm does not to create candidate set $\{I_1, I_2, I_3\}$. Now, the algorithm jumps to another child of I_2, i.e. I_4. We can see that $deg(I_4) = 2$ but the heuristic 2 tells the algorithm that the set of incoming nodes of I_4 $\{I_2, I_3\}$ is not a superset of the set of visited nodes $\{I_1,I_2\}$, hence the algorithm does not have to consider the set $\{I_1,I_2,I_4\}$. The algorithm then starts other iterations on vertices I_2, I_3, I_4 and ends up finding other frequent sets $\{I_2, I_3\}$, $\{I_2, I_4\}$ and $\{I_3, I_4\}$.

5.4. Data Preprocessing

In order to improve the performance of the algorithm, it is necessary to clean the data before running the algorithm. The cleansing preprocessing stage will eliminate all attributes columns that are infrequent as well as all instances that do not contain any frequent attributes.

5.5. Mining Parametric Definition of Phrases.

Note that, since we extract data from the Web by posing a search phrase query to a web search engine, all the instances in the data we get contain search phrase. Therefore, the association rule generation becomes simple by just putting the search phrase into the header of association rules and the body of rules is frequent itemsets. The support of obtained association rules equals to the support of frequent items set in their body since for a rule, the search phrase occurs in all instances that the frequent itemset (in the body of the rule) occurs.

Next, we would like to utilize the extracted product information to mine parametric phrase definition rules made up from conjunctions of distinct <attribute, value> pairs, like:

Trendy shoe ←
brand = Steve Madden,
Color = black,
material = leather (1)

Let us consider the following extracted sample database

ID	Brand	Style	Color	Material
01	Paul Green	oxford	Black	leather
05	Sesto Meucci	moc toe	Black	leather

Figure 5. Database II

b_1	b_2	s_1	s_2	c_1	m_1
1	0	1	0	1	1
0	1	0	1	1	1

Figure 6. Transformed Database of Database II

In order to facilitate the frequent itemset mining algorithm to mine frequent sets in the form of (1), we introduce a simple transformation to convert the database in Figure 13 into the form shown above in Figure 14.

The Database II and its transformed database are called *frequent itemset equivalent* databases. That means each frequent itemset of database II is equivalent to a frequent itemset in its transformed database. These two databases have the same number of instances but the number of items in transformed database is much larger than the number of items in original database. Specifically, the number of items in transformed database is a polynomial of the number of items in the original database.

After the transformation, we can apply the frequent itemset mining algorithm to transformed database in Figure 14 to mine the frequent itemsets in the form (1) for the database in Figure 13. For the numeric parametric items, we adopt a discretization method to classify the values into discrete intervals to facilitate the mining algorithm. For example, for Price we have intervals C_1=[0-$20], C_2 = [$21-$40] and so on. Furthermore, we observe that those items in the database in Figure 13 that fall into the same attribute class, for example, items m_1, m_2 and m_3 never occur pair-wise in the same frequent itemset. Therefore, we group items corresponding to the same attribute and during the construction of 2-frequent itemset graph, the graph never contains edges that connect any pair of items that fall into the same attribute group. This constraint reduces the search space and enhances the performance.

5.6. Mining Textual Definitions of Target Phrases

Another resource of rich phrase definitions is the long product descriptions of the matching products. In the Section 4, we have already described how we plan to collect long product descriptions from product Web pages that matches a given target search phrase. In this section we describe the proposed algorithm for mining phrase definitions that can connect hidden phrases to product descriptions themselves. An example of a long product description that matches "trendy shoe" is**:**

"*Get celebrity elegance, stylish look and luxury all in one with these Susan Lucci Suede Pumps with Lace-up Detail. You can choose from black or camel. The shoe's upper features suede and smooth leather with lacing detail at the center vamp. The lacing extends from the center top of the vamp to the side quarters. The pumps have a snip toe and rounded throat line. These shoes also have a manmade ribbed outsole to resist slippage and an approx. 3-1/4"H suede covered heel. Other features include: Susan Lucci TM couture-like, trendy apparel is inspired by her own personal collection and is designed for the fashion savvy woman who truly appreciates Hollywood glamour and style. Incorporating silk blends with stretch fabrics, her fashions give you the look of glamour with comfort and ease*"

One can already identify phrases, such as "celebrity elegance", "stylish look", "Suede Pumps", "Lace-up Detail", "smooth leather", "fashion savvy woman" that could indicate "trendy shoe" status for a product. Hence, our first step is to identify frequently occurring phrases within long descriptions matching a hidden phrase. In order to generate candidate phrases first we perform part-of-speech (POS) tagging and noun and verb phrase chunking [20] on the long description to obtain a more structured textual description. Part-of-speech (POS) tagging and chunking the above description yields the following structure.

The phrases in between ([…]) corresponds to noun phrases. Next, we generate candidate words, and phrases of length two words, three words ect from the noun phrases in the matching long descriptions. For the above example this would yield a list of words including "celebrity", "elegance", "stylish" etc, two words phrases such as "celebrity elegance", "stylish look", "couture-like apparel", "trendy apparel", and similarly three word phrases such as "couture-like trendy apparel", "fashion savvy woman" etc.

```
- <TEXT>
  - <P>
      <S>((Get )) ([ celebrity elegance ]), ([stylish
         look]) and ([ luxury ]) all in one with ([
         these Susan Lucci Suede Pumps ]) with
         ([ Lace-up Detail ]).</S>
      <S> …
      <S>([The shoe 's upper features suede ]) and
         ([ smooth leather ]) with ([ lacing detail
         ]) at ([ the center vamp ]).</S>
      <S> …
      <S>([Other features ]) (( include )): ([ Susan
         Lucci ]) ([ s ]) ([ couture-like , trendy
         apparel ]) (( is inspired )) by ([ her own
         personal collection ]) and (( is designed
         )) for ([ the fashion savvy woman ]) who
         (( truly appreciates )) ([ Hollywood
         glamour ]) and ([ style ]).</S>
      <S> …
    </P>
  </TEXT>
```

In the next step, we utilize the noun phrases as transaction instances and mine frequently used phrases from all the noun phrases of all the product descriptions that we have collected from the Web documents. This step yields frequently used phrases such as "stylish look" and eliminates not-so-frequent phrases such as "her own personal collection". Notice that the frequent phrases generated in this step is richer than frequent bi-gram, or n-gram models since we can create candidate phrases by selecting non-consecutive words such as "couture-like apparel" in the candidate phrase generation step.

Next, we use the mined frequent phrases as items and create transaction instances by marking all of the frequently used phrases matching anywhere in the long description. This would yield transaction instances made-up from frequently used phrases matching the product descriptions.
Example: The above long description might yield the following transaction instance:
{"celebrity", "elegance", "celebrity elegance", "luxury", "suede", "pump", "suede pump", "lace-up", "detail", "lace-up detail", "smooth", "leather", "smooth leather", "fashion", "savvy", "woman", "fashion savvy", "savvy woman", "fashion savvy woman"}

Next we mine the frequent itemsets among instances corresponding to the long descriptions to find the phrase definitions. Note that, due to our way to construct the items, all items are combinations of single words; therefore, there are items that subsume other items. As a subsequence, there are a lot of redundant final resultant frequent itemsets. For example a long description might yield the following items: "suede", "pump", "suede pump", "fashion", "savvy", "woman", "fashion savvy", "savvy woman", "fashion savvy woman". Hence, we only want to mine the frequent itemset "suede pump", "fashion savvy woman" because these frequent itemsets subsume the former frequent itemsets. The frequent phrases by construction form a lattice.

In order to prune the redundant frequent itemsets and also to improve the performance of textual definition mining process, we integrated into the frequent itemset mining algorithm the component that makes use of the above lattice structure and visits the items by utilizing the partial order in the lattice, from larger to smaller phrases. The component will traverse top-down the lattice. If there are two items that form a 2-frequent itemset and one subsumes the other (one item is a superset of the other) for example phrase "*fashion savvy woman* " subsumes phrase "*savvy woman* ", then in the 2-frequent itemset graph does not contain an edge that connect these two items. This pruning technique avoids producing redundant frequent itemsets thus improving the performance of the long description miner.

6. Experimental Results

In this section, we show the experimental results for several target phrases from three different product categories. The extraction engine and the frequent itemset mining algorithms that we develop in previous sections are used to perform the experiments for the following product categories: *shoes*, *handbags* and *beddings*, for the target phrases: *discount shoes, trendy shoes, fashion handbags, luxury bedding* and *sport beddings.* After extracting parametric tabular data and long descriptions from Web sites for each of the target phrases, we used parametric frequent itemset mining algorithm and textual frequent item set mining algorithm to mine the phrase definitions. The tables below show some of the definitions that were mined. It is a relatively easy task for a domain expert to inspect and evaluate the quality of such rule-based definitions.

6.1. Comparison to Relevance Feedback Method

In order to compare the performance of our definition miner to standard relevance feedback retrieval method

	Parametric Rules	**Support**
Fashion handbags	Brand = Jil Sander, material = leather, type = clutch ➔ fashion handbags	4.25%
	Brand = Carla, design = mancini, material = leather ➔ fashion handbags	2.4%
	Brand = Butterfly, design =beaded ➔ fashion handbags	2.4%
	Brand = Sven, material = leather ➔ fashion handbags	10.2%
	Design = beaded, color = pink ➔ fashion handbags	2%
	Design = beaded, color = blue, type = tote ➔ fashion handbags	3.2%

	Parametric Rules	**Support**
Luxury beddings	Design = Baffled box, material = cotton ➔ luxury beddings	5%
	Design = Waterford, material = linen ➔ luxury beddings	6%
	Material = silk ➔ luxury beddings	3%
	Design = Sussex, material = polyester ➔ luxury beddings	6%
Sport beddings	Design = All American, material = polyester ➔ sport beddings	6%
	Design = All star, material = polyester ➔ sport beddings	9%
	Design = Big and bold ➔ sport beddings	17%
	Design = sports fan ➔ sport beddings	45%

	Textual Rules	**Support**
Luxury beddings	Satin, embroidery ➔ luxury beddings	1.1%
	Egyptian cotton mate-lass ➔ luxury beddings	0.6%
	Silk, smooth, King set ➔ luxury beddings	0.75%
	Piece ensemble ➔ luxury beddings	0.75%
Sport beddings	Addition pillow ➔ sport beddings	0.4%
	American sport ensemble ➔ sport beddings	0.4%
	Paraphernalia sport ➔ sport beddings	0.6%

	Textual Rules	**Support**
Trendy shoes	casual, leather ➔ trendy shoes	6%
	fashionable sneaker ➔ trendy shoes	7%
	Wedge edge ➔ trendy shoes	5%
	Platform shoes ➔ trendy shoes	6%

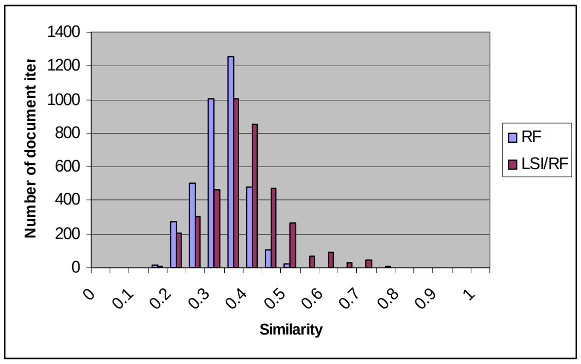


Figure 7. Similarity histogram for relevance feedback and relevance feedback with LSI

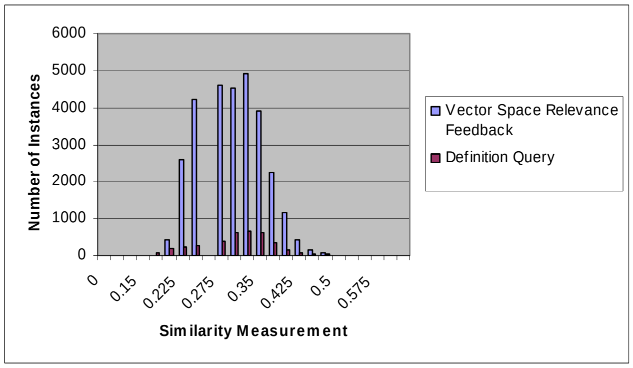


Figure 8. Similarity histogram for rule-based and relevance feedback based matches

we mined a large database of shoes (33,000 items) from a collection of online vendors. Next, we keyword queried the database with the target exemplary search phrase “trendy shoe”.From the 166 keyword matching shoes, we mined rule-based phrase definitions for “trendy shoes” yielding rules such as fashionable sneaker, platform shoes etc. that were validated by a domain expert. These mined rules matched 3,653 additional shoes. Alternatively, we also computed the relevance feedback query vector using the above 166 matching shoes. We also identified a similarity threshold by finding the maximal cosine theta, Θ, between the relevance feedback query vector and all of the 166 shoe vectors. Retrieval using the relevance feedback vector with this threshold yields more than 29,000 matches out of 33,000! The light colored bars in Figure 8 above illustrate the histogram plot of the 29,293 instances that falls into various similarity ranges. Similarly, the dark colored bars plots the similarity ranges of the 3,653 shoes that were retrieved by matching with our mined definitions. As can be seen from the distributions in the above chart, the items retrieved with our mined definitions have a very uniform similarity distribution (with around 300 of these being below the threshold), as opposed to having a skewed distribution towards the higher values of similarity. Since dark colored bars correspond to relevant “trendy shoes” matching our rules, which were validated by an expert, most of these items should have ranked towards the higher end of the similarity spectrum. However, relevance feedback measure failed to rank them as such; hence, it performed poorly for this task.

6.2. Comparison to Relevance Feedback with LSI

The plot of similarity ranges obtained by ranking the 3,653 shoes, retrieved with our mined rules, using relevance feedback with and without latent semantic indexing (LSI) [25] technique is shown in Figure 7. The light colored dashed line represents the cosine theta threshold Θ for the relevance feedback ranking, similarly the dark colored dashed line represents the cosine theta threshold for the relevance feedback with LSI. The recall for relevance feedback is nearly 93%, however, since it matches 88% of a random collection of shoes, its precision is lower. On the other hand, even though the ranking of relevance feedback with LSI falls onto a higher similarity range, it appears to have a much lower recall (of 25%) for this experiment with exemplary target phrase “trendy shoes”.

7. Conclusions and Future Work

Our initial experimental results for mining phrase definitions are promising according to our retail domain expert who is the Webmaster of an affiliate marketing web site. We plan to scale up our experiments to hundreds of product categories and thousands of phrases. Also, we would like to perform experiments to determine how precisely our algorithm learns the definitions of phrases that changes their meaning over time.

References

[1] R. Agrawal and R. Srikant.: “Fast Algorithms for mining association rules”. *In Proc. 20th Int. Conf. VLDB* (1994) 487-499

[2] H. Aholen, O. Heinonen, M. Klemettinen, and A. I. Verkamo.: “Applying Data Mining Techniques for Descriptive Phrase Extraction in Digital Collections”. *In Proceedings of ADL’98*, Santa Barabara, USA

[3] W. Andrews, “Gartner Report: Visionaries Invade the 2003 Search Engine Magic Quadrant”, April 2003.

[4] V. Crescenzi, G. Mecca, and P. Merialdo, “Roadrunner:Towards automatic data extraction from large web sites”, *In Proc. of the 2001 Intl. Conf. on Very Large Data Bases*, 2001.

[5] Cutting and R. Douglas.: Real life information retrieval: Commercial search engines. Part of a panel discussion at SIGIR 1997: *Proc. of the 20th Annual ACM SIGIR Conference on Research and Development on Information Retrieval* (1997)

[6] B. Goethals, “Survey on Frequent Pattern Mining”, Department of Computer Science, University of Helsinki, Finland. Available at: www.cs.helsinki.fi/u/goethals/publications/survey.pdf

[7] J. Hammer, H. Garcia-Molina, J. Cho, A. Crespo, and R. Aranha, “Extracting semi-structure information from the web”, *In Proceedings of the Workshop on Management of Semistructured Data*,1997.

[8] J. Han J.Pei, Y.Yin, and R. Mao. “Mining frequent pattern without candidate generation.” *In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, volume 29(2) of SIGMOD Record, ACM Press, 2000.

[9] J. Han, and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, 2001

[10] J. Karlgren.: Non-topical factors in information access. Invited talk at WebNet '99, Honolulu, Hawaii, USA, (10,1999)

[11] M.F.Porter. An algorithm for suffix stripping, Program, 14 no. 3, pp 130-137, July 1980.

[12] N. Kushmerick, D. Weld, and R. Doorenbos, “Wrapper induction for information extraction”, *In Proc. of the 1997 Intl. Joint Conf. on Artificial Intelligence*, 1997.

[13] B. Len, R.Agrawal, and R. Srikant.: “Discovering trends in text databases”. In D. Heckerman, H. Mannila,D. Pregibon, and R. Uthrysamy, editors, *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining (KDD’97),* Newport Beach, California, USA (8,1997). AAAI Press 227-230

[14] Y. K. Liu.: Finding Description of Definitions of Words on the WWW. Master thesis, University of Sheffield, England, 2000.

http://dis.shef.ac.uk/mark/cv/publications/dissertations/Liu2000.pdf

[15] Hung V. Nguyen, P. Velamuru, D. Kolippakkam, H. Davulcu, H. Liu, and M. Ates. Mining "Hidden Phrase" Definitions from the Web. *APWeb* 2003, 23-25, April 2003, Xi'an, China.

[16] M.F.Porter, “An algorithm for suffix stripping”, *Program*, 14 no. 3, pp 130-137, July 1980.

[17] M.J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372-390, May/June 2000.

[18] B. Chidlovskii. Automatic repairing of web wrappers. *WIDM* 2001.

[19] I. H. Witten, E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, October 1999.

[20] Steve Finch and Andrei Mikheev. A Workbench for Finding Structure in Texts. *Applied Natural Language Processing* , Washington D.C., April 1997.

[21] Ellen M. Voorhees. Using WordNet for Text Retrieval. In WordNet: An Electronic Lexical Database, Edited by Christiane Fellbaum, MIT Press, May 1998.

[22] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. Journal of the American Society for Information Science, pages 288--297, 1990.

[23] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. Modern Information Retrieval, ACM Press / Addison-Wesley, 1999.

[24] H. Davulcu, S. Vadrevu, S. Nagarajan, I.V. Ramakrishnan. OntoMiner: Bootstrapping and Populating Ontologies From Domain Specific Web Sites. In *IEEE Intelligent Systems*, Volume 18, Number 5 September/October 2003.

[25] Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R. A. Indexing. Latent semantic analysis. Journal of the Society for Information Science, 1990, 41(6), 391-407.