Topical Presentation of Search Results on Database*

Hao Hu[1,2], Mingxi Zhang[1,2], Zhenying He[1,2], Peng Wang[1,2], Wei Wang[1,2], and Chengfei Liu[3]

[1] School of Computer Science, Fudan University, Shanghai, China
[2] Shanghai Key Laboratory of Data Science, Fudan University
[3] Faculty of ICT, Swinburne University of Technology, Melbourne, Australia
{huhao,10110240025,zhenying,pengwang5,weiwang1}@fudan.edu.cn,
cliu@swin.edu.au

Abstract. Clustering and faceting are two ways of presenting search results in database. Clustering shows the summary of the answer space by grouping similar results. However, clusters are not self-explanatory, thus users cannot clearly identify what can be found inside each cluster. On the other hand, faceting groups results by labelling, but there might be too many facets that overwhelm users.

In this paper, we propose a novel approach, topical presentation, to better present the search results. We reckon that an effective presentation technique should be able to cluster results into reasonable number of groups with intelligible meaning, and provide as much information as possible on the first screen. We define and study the presentation properties first, and then propose efficient algorithms to provide real time presentation. Extensive experiments on real datasets show the effectiveness and efficiency of the proposed method.

1 Introduction

Database query results can be presented in a ranked list, in clusters or in facets. Ranked list is popular; however, it does not help navigate the answer space. In contrast, clustering and faceting are designed for users to quickly get a general picture of the whole answer space first, and then to locate relevant results. However, several issues remain to be studied. Let us first consider a simple example.

Example 1. *Consider a laptop database that maintains information like Brand, Screen Size, CPU type, etc. First, assume the results are presented in clusters. Table 1 shows a typical interface, similar laptops are grouped into the same cluster, and can be accessed by hyperlink in* Zoom-in *field. There are two shortcomings. (1) Clusters may be ambiguous (i.e., not self-explanatory). Users could be puzzled of what can be*

* This work was supported in part by NSFC grants (61170007, 60673133, 61033010, 61103009), the Key Project of Shanghai Municipal Science and Technology Commission (Scientific Innovation Act Plan, Grant No.13511504804), and ARC discovery project DP110102407.

S.S. Bhowmick et al. (Eds.): DASFAA 2014, Part II, LNCS 8422, pp. 343–360, 2014.

Table 1. Clustering example

ID	Brand	Price	Color	CPU	ScreenSize	*Zoom-in*
05	Dell	539	Black	Intel i3	14	311 more laptops like this
06	Apple	1499	Silver	Intel i5	13	217 more laptops like this
07	HP	559	Black	Intel i7	13.3	87 more laptops like this
08	Sony	729	Pink	Intel i3	14	65 more laptops like this

Table 2. Results after applying *Zoom-in*

Brand	Price	Color	CPU	...
Dell	539	Black	Intel i5	...
HP	549	Black	Intel i7	...
Dell	559	Silver	Intel i5	...
HP	559	Pink	Intel i3	...

found through the hyperlink. We refer to this as ***S1*** *(shortcoming 1, unknown label). (2) For thousands of results (****S2****, result overwhelming), if the number of clusters k is set small, each cluster may also contain many results, this might lead to other issues.*

(i) *Presenting each cluster may overwhelm users.*

(ii) *Results in one cluster may not be similar to each other. Table 2 shows the results after applying* Zoom-in *to the first cluster in Table 1. These laptops are considered* similar *according to the distance measure of the system. However, in fact, a novice user might think* Dell *and* HP *are* similar *for some reasons such as the prices are nearly the same, while a professional user may consider them quite different. This is a negative effect of* ***S1****, as the semantic similarity is unknown to users.*

Second, assume the results are presented by faceting, such as `amazon.com`*. Facets (i.e., labels) are listed on the left side of each returned web page. Actually, when submitting "laptop" as keyword, there are more than 100 facets, which cannot be fulfilled on the first screen. Each facet represents a group of results, and too many facets may lead to dissatisfaction as well (****S3****, facets overwhelming).* ■

Clustering solves the result overwhelming issue (**S2**) by setting a fixed number of clusters. However, each cluster is not self-explanatory. On the other hand, faceting groups results with labels; nevertheless, it might overwhelm users (**S3**).

In this paper, we try to combine the advantages of both sides. However, straightforward solutions may not be applicable.

On one hand, directly adopting label extraction methods [17,20] for clustering is improper. First, most label extraction methods are designed for document sets, while these methods are effective for texts, we cannot make the most of them for structured data. Second, label extraction methods often employ supervised learning algorithms, they emphasize the quality. However, result presentation requires real time interaction in practice.

On the other hand, selecting a few facets to avoid overwhelming may deliver insufficient information. Most of the facet selection methods [12,6] only consider result size, i.e., the goal is to select k facet groups that maximize the total result size. In the `amazon` example, top-3 facets are "windows 7", "windows 8", and "windows vista". Though too large facets could cover more tuples, they deliver not sufficient information on the first screen.

In this paper, we study how to present search results on database. Specifically, the goals that a good presentation approach should meet are as follows.

G1. (Goal 1, on **S1**) Each grouped results (packages) should be intelligible (i.e., packages should have labels), hence users can understand it easily.

G2. (On **S2**) Reasonable size of each package should be ensured, thus users are not overwhelmed. We do not prefer packages with too many tuples (overwhelming users) or too few tuples (providing insufficient information).

G3. (On **S3**) Due to the first screen size, only k packages can be presented, we should make these summarized k packages bring as much information as possible, so as to give users a more informative picture of the result set.

We propose a topical presentation (TP) approach for these goals. TP takes query result as input, and outputs k intelligible packages, where each package is neither too large nor too small, and collectively, the k packages aim to provide maximal information. To this end, we first need to generate all packages, and then summarize them.

We have identified three challenges. **First**, there are many packages in practice, we need to efficiently generate them and then summarize them. **Second**, the information of each package is hard to measure. Entropy-based measures are well studied in information theory. However, it is inappropriate to adopt these measures for packages with labels, because defining row wise entropy or defining the distribution of tuples is hard. Another difference is that entropy-based methods measure the information of the package, whereas we need to measure the information of both the package and its labels. **Third**, the k packages might overlap. For G3, TP needs to maximize the information without the overlapped tuples. Intuitively, the goal of summarizing packages is to expose users (1) as many tuples as possible and (2) as many labels as possible. However, the two aspects are inversely proportional (e.g., a facet with label "windows 8" consists more laptops than a facet with labels "windows 8" and "DELL").

To tackle the challenges, TP first analyzes how to get labelled groups (G1), then determines the acceptance for groups to achieve a reasonable group size (G2). Next, we introduce a metric to measure the information of each group. Finally, a fast algorithm is proposed to select k groups with maximal information (G3). We achieve this goal based on the *maximizing k-set coverage* principle.

For Example 1, four intelligible groups are shown in Table 3. We choose a representative tuple for each group. Each group is assigned with several labels (highlighted in boxes) to describe the commonality of tuples in it. These groups help users to learn the answer space from different aspects. We aim to provide maximal information by choosing most representative labels in these groups.

Table 3. Topical presentation example

ID	Brand	Color	CPU	ScreenSize		*Subtopic*		
01	Dell	Black	Intel i5	14	186	Dell	Black	laptops
02	Sony	Pink	Intel i5	15	17	Sony	15 inches	laptops
03	Lenovo	Black	Intel i5	14	168	Intel i5	14 inches	laptops
04	MacBook	Silver	Intel i7	14	66	Silver	MacOS	laptops

Contributions. (1) We propose and define the TP problem. (2) We propose a metric to measure the information of each package. (3) We suggest a novel mechanism for summarizing many packages into k, and propose a fast algorithm. (4) Extensive experiments are performed to evaluate the proposed approach.

Roadmap. Section 2 defines the problem (G1). Section 3 generates acceptable packages (G2). In Section 4, we summarize the packages (G3). Section 5 shows the experiments, Section 6 and Section 7 discuss the related work and conclusion.

2 Problem Definition

We adopt the labels used in faceting to deliver clear meaning. Labels are from attribute values of tuples. This section begins with preliminaries, and then defines the problem. The terminology in OLAP is used. Table 4 lists some notations.

Table 4. Notations used in this paper

Symbol	Description	Symbol	Description	Symbol	Description
T	relation	l	label	$\mathcal{D}$	attribute space
mt	meta-topic	$OS(P_{st})$	overall score	$o(P_{st})$	overview ability
st	subtopic	$OS_m(\mathcal{P})$	informative score	$m(P_{st})$	meaningfulness
$\pi(mt)$	partition of mt	$\mathcal{M}$	set of packages that are from one mt	$sup(P_{st})$	support of P_{st}
P_{st}	package of st	I_C	summary set		
$\mathcal{P}$	package set	R	query result set	$cha(t,l)$	character for label l in t

2.1 Preliminaries

Definition 1 (Meta-topic). *Let T=(A_1,...,A_n) be a relation with attributes A_i. A* **meta-topic** *on T is a combination of attributes: mt=(x_1,...,x_n) where x_i=A_i or x_i=$*$ ($1\leq i\leq n$), and $*$ is a meta symbol meaning that the attribute is generalized. The* **partition** *of mt (denoted as $\pi(mt)$) is a set of tuple sets, where each tuple set consists of tuples with same values on non-$*$ attributes of mt, i.e.,*

$$\pi(mt)=\{s|\forall t_i,t_j\in s{:}t_i[v_k]=t_j[v_k] \text{ if } x_k\neq *,1\leq k\leq n\} \quad (1)$$

where $t_i[v_k]$ denotes the value of attribute A_k in tuple t_i.

For mt=(x_1,...,x_n) and mt'=(y_1,...,y_n), mt is an ancestor of mt' (i.e., $mt\succ mt'$) if x_i=y_i for each $x_i\neq *$, and there exists $j\in[1,n]$ such that x_j=$*$ but $y_j\neq *$. ■

Take Table 3 as an example, assume there are only 4 attributes, T = (*Brand*, *Color*, *CPU*, *ScreenSize*), the partition of meta-topic ($*$,*Color*,*CPU*,$*$) contains three tuple sets (we use IDs in Table 3 to represent each tuple), {01,03},{02},{04}. Moreover, ($*$,$*$, *CPU*,$*$) $\succ$ ($*$,*Color*, *CPU*, $*$).

Corollary 1 (Refinement). *Given mt, mt', if $mt'\succeq mt$, then $\pi(mt)$ refines $\pi(mt')$, i.e., $\forall s\in\pi(mt),\exists s'\in\pi(mt'){:}s\subseteq s'$.* ■

In Table 3, tuple sets {{01,03},{02},{04}} refines {{01,02,03},{04}}, where the latter is the partition of ($*$,$*$,*CPU*,$*$). For tuple set {01,02,03}, tuples share common label `Intel i5`. All common labels of a tuple set consist of its subtopic.

Definition 2 (Subtopic). *Given a relation T, and a meta-topic mt=(x_1,...,x_n), a* **subtopic** *for mt, denoted as $st\in mt$, is a combination of attribute values: st=(z_1,...,z_n) where $z_i\in A_i$ (if x_i=A_i) or z_i=$*$ (if x_i=$*$). We call each non-$*$ z_i a* **label** *of st, and refer to all tuples of a subtopic st as a* **package** *(denoted as P_{st}), i.e.,*

$$P_{st}=\{t|t\in T,t[v_k]=z_k \text{ if } z_k\neq *,1\leq k\leq n\} \quad (2)$$

The **support** *of P_{st} is defined as $sup(P_{st})={}^{|P_{st}|}/_{|T|}$. Packages of all subtopics consist of $\pi(mt)$, if all the subtopics are from mt. i.e., $\cup_{\forall st\in mt}P_{st}=\pi(mt)$.* ■

For example, ($*$,$*$,`Intel i5`,$*$) is a subtopic, it is an instance of meta-topic ($*$,$*$,*CPU*,$*$). The package of this subtopic is $P_{(*,*,\texttt{Intel i5},*)}$ = {01,02,03}. If there are only 4 tuples in T, its *support* is 0.75.

In this paper, we assume all tuples are in one table and attribute values are categorical. For numeric data, we assume it has been suitably discretized.

A subtopic can also be viewed as a label set. We use notations in set theory, such as $|st|$, $st_1 \cup st_2$, $st_1 \cap st_2$, to denote the number of labels in st, all distinct labels, and labels belonging to both st_1 and st_2, respectively.

2.2 Overview Ability and Meaningfulness

It is obvious that the more labels describing a package (more meaningful), the less tuples it contains (less overview ability). This is a trade-off between (1) the overview ability of the answer space; and (2) the meaningfulness of each package.

Overview Ability. It is easy to see that, a package with more tuples has better overview ability. Therefore, the overview ability of P_{st} is defined as the proportion of the whole answer space in this paper.

$$o(P_{st}) = \Sigma_{t \in P_{st}} score(t) / \Sigma_{t \in R} score(t) \tag{3}$$

where R is the query result set, and $score(t)$ is an adaptation of existing scoring techniques. For example, $score(t)$ can be the relevance between keyword query Q and t, or the feedback of t by users.

Meaningfulness. As discussed in Section 1, it is hard to quantify the meaningfulness. To this end, we assume that, a package P_{st} is more meaningful if there are more labels in st. This is a natural assumption because users are exposed with subtopics directly; they can learn more information if and only if there are more labels. The meaningfulness is defined as follows.

$$m(P_{st}) = \Sigma_{l \in st} weight(l) \tag{4}$$

where $l \in st$ is a label, $weight(l)$ measures the importance of l. Many approaches (e.g., query log mining, frequency based scoring) have been proposed for label scoring. We assume the labels are independent for simplicity, hence the meaningfulness can be denoted as the sum of all label weights.

Overall Score. The overall score for overview ability and meaningfulness of a package is defined as follows.

$$OS(P_{st}) = o(P_{st}) \times m(P_{st}) \tag{5}$$

We use the product of $o(P_{st})$ and $m(P_{st})$ because the two aspects are in inverse proportion. For a package, we prefer high score of both $o(P_{st})$ and $m(P_{st})$.

Remark 1. Note that $OS(P_{st})$ can be applied to many package ranking methods. For example, for frequency-based ranking [6] (where a facet/package is ranked by the number of tuples in it), we could set $score(t)$=1 and $m(P_{st})$=1. As a result, the ranking by $OS(P_{st})$ is equivalent to the frequency-based ranking.

Remark 2. There are many implementations for $score(t)$ and $weight(l)$, e.g., TFIDF, user feedback. Different scenarios require different scoring functions, thus we leave $score(t)$ and $weight(l)$ untouched, to make $OS(P_{st})$ adaptive.

Remark 3. Substituted by Eq. 3 and 4, $OS(P_{st}) = \sum_{t \in P_{st}} \sum_{l \in st} score(t) weight(l) / \sum_{t \in R} score(t) \propto \sum_{t \in P_{st}, l \in st} score(t) weight(l)$. Denote $cha(t,l) = score(t) weight(l)$ as the *character* for label l in tuple t, the overall score can be rewritten as $OS(P_{st}) \propto \sum_{t \in P_{st}, l \in st} cha(t,l)$.

2.3 Problem Definition

Problem 1 (TP). Given search results R, attribute space $\mathcal{D}=\cup_{i=1}^{|\mathcal{D}|}A_i$ and an integer k. Let $OS_m(\cdot)$ be the informative score of multiple packages, and $\mathcal{P}$ be all packages on $\mathcal{D}$, the topical presentation (TP) problem is to select a summary set I_C, that

$$\max_{I_C\subseteq\mathcal{P},|I_C|=k} OS_m(I_C), s.t. \forall P_{st}\in I_C \begin{cases} P_{st} \text{ is acceptable} \\ st \text{ is on } \mathcal{D} \end{cases}$$

where $OS_m(I_C)=f(\cup_{P_{st}\in I_C}OS(P_{st}))$ measures the information of all packages in I_C.

Following the previous example, in frequency-based ranking, $OS_m(I_C) = \sum_{P_{st}\in I_C}OS(P_{st})$, and in set cover ranking, $OS_m(I_C) = |\cup_{P_{st}\in I_C} P_{st}|$.

In Problem 1, each $P_{st}\in I_C$ is required to be acceptable due to **G2**, and $OS_m(I_C)$ is required to be maximal due to **G3**. To compute I_C, we need all packages $\mathcal{P}$, informative score $OS_m(\cdot)$, and a fast algorithm. These issues are discussed next.

3 Package Generation

Given search results R and attribute space $\mathcal{D}$, by the definition of subtopic, we can use the *group-by* in SQL to generate packages. The DBMS could return all packages of a meta-topic mt, if we *group-by* all non-$*$ values in mt. However, it needs to execute the *group-by* $2^{|\mathcal{D}|}$ times, since there are $2^{|\mathcal{D}|}$ meta-topics on $\mathcal{D}$. This is time consuming. Besides, not all packages are interesting, consider a package with st = (`DELL`,`Pink`, `Intel i5`,`13inches`), actually, there is only one laptop in this package, thus presenting it on the first screen might not be desired.

We now describe acceptable packages and the package generation algorithm.

3.1 Acceptable Package

An acceptable package is valid, and has neither too many nor too few tuples.

Valid Packages Due to the *finer-than* relation between meta-topics (Corollary 1), if $\pi(mt_1)$ refines $\pi(mt_2)$, then the partition of their *meet* (a partition refines both mt_1 and mt_2), $\pi(mt_1 \wedge mt_2)$ is equivalent to $\pi(mt_1)$. In order to eliminate the redundancy, we propose the notion of valid package.

Definition 3 (Valid Package). *A package P_{st}, $st\in mt$, is valid if there is no descendant mt' of mt(i.e., more specific than mt) such that $P_{st'\in mt'}=P_{st}$.* ■

The intuition of valid package is that if two packages are same in tuples, but different in subtopics, e.g., (∗,`Pink`,`Intel i5`,∗) and (∗,`Pink`,`Intel i5`,`13inches`). We prefer the latter package, for it has more meaning.

Example 2 *The* finer-than *relation of meta-topics is a partial order and it is a complete lattice. Fig. 1 shows a table T and the lattice. $\pi((A,*,*))$ refines $\pi((*,B,*))$, thus the partition of their* meet *$\pi((A,B,*))$, is the same with $\pi((A,*,*))$. Therefore, packages from $\pi((A,*,*))$ are invalid. Besides, packages from $\pi((A,*,C))$ and $\pi((*,*,C))$ are also invalid.*

Table T

ID	A	B	C	D, ...
1	a_1	b_1	c_1	w_1, w_2
2	a_1	b_1	c_2	w_1, w_2
3	a_2	b_2	c_3	w_1, w_2
4	a_2	b_2	c_3	w_1, w_2
5	a_3	b_2	c_3	w_1, w_2

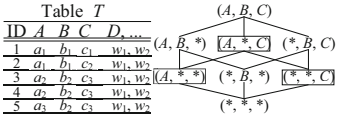


Fig. 1. Illustration on Lattice, the attribute space $\mathcal{D} = ABC$

Package Support Threshold. For packages with too many or too few tuples, we need to determine the maximal and minimal support (denoted by *min_sup* and *max_sup*) as thresholds. The thresholds can be set by system administrators according to experience, or by users. One can easily determine *min_sup* by setting a minimal package size (e.g., 5, 10, or 15). However, determining *max_sup* is hard, it depends on the result size $|R|$ and the distribution of tuples.

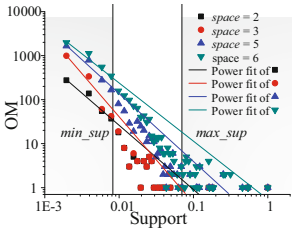


Fig. 2. *Zipf* distribution

In this work, we report that for real dataset, the support is described by a *Zipf* distribution over the meaningfulness of packages. Fig. 2 shows the meaningfulness is inversely proportional to the support. The packages are generated from 511 laptops with 11 attributes, and for all packages with the same support, we sum their meaningfulness as overall meaning, i.e., $OM(s)=\sum_{sup(P_{st})=s} m(P_{st})$. We can see that *Zipf* distribution is obeyed for different attribute space. Therefore, following [11], we estimate *max_sup* according to the transition point (denoted as *tp*) calculation of *Zipf*'s second law.

$$OM(tp) = \frac{1}{2}\left\{-1 + \sqrt{1 + 8count(OM(1/|R|))}\right\} \tag{6}$$

where $count(OM(1/|R|))$ counts the number of packages with support $1/|R|$. This equation is adapted from [8]. Several support values s may satisfy $OM(s)=OM(tp)$, and we choose a minimal one as *max_sup*.

Remark 4. Determining the threshold may have other choices. For example, we can set an overall score *min_os* and mark packages with $OS(P_{st}) \leq min\_os$ unacceptable. However, it is hard to determine *min_os*. Extensive turning work needs to be done for a better *min_os*.

3.2 An Apriori Style Approach

A naive package generation method generates all packages first and then removes unacceptable ones. However, this `BaselineGeneration` algorithm is inefficient.

Algorithm 1. `FastPackageGeneration`

Input : Query result set R, attribute space $\mathcal{D}$
Output: all acceptable packages
1 let $Mt_{(1)}=\{mt_1,mt_2,...,mt_{|D|}\}$ be 1-size meta-topics constructed from $\mathcal{D}$;
2 **for** k=1 *to* $|D|-1$ **do**
3 generate packages from each meta-topic in $Mt_{(k)}$;
4 remove invalid and too small packages, mark too big packages as "removed";
5 generate k+1-size meta-topics $Mt_{(k+1)}$ with $Mt_{(k)}$;

Algorithm 1 (`FG` for short) is a fast generation algorithm using the valid and threshold conditions for filtering. To find invalid packages as soon as possible, we iterate meta-topics level-wise, where k-size meta-topics are used to explore k+1-size meta-topics. When iterating, we check the thresholds and the validity to avoid generating the unacceptable packages (Line 4), thus the searching space is reduced. When storing packages in each meta-topic, we use a heap to keep them ordered by overall score.

4 Summarization

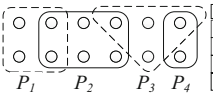
We now define the summarization goal $OS_m(\cdot)$ and describe the algorithms.

4.1 Summarization Goal

Clustering is commonly adopted for summarization. We can define a distance measure between packages, and then cluster all packages into k clusters. However, it is hard to define a distance, e.g., Jaccard distance cannot tell the difference between two packages from one meta-topic, since there are no overlapped tuples.

This paper explores a different approach by leveraging the principle of *maximizing k-set coverage*. Specifically, we consider the goal of summarization as the following: maximizing overview ability and meaningfulness. Intuitively, this provides users the best balance between overview and understanding of summaries.

This principle is better illustrated in Fig. 3. Assume we want to pick two packages out of the four total packages (e.g., k=2). Selecting P_2 and P_3 allows users to view 9 of 12 tuples, and learn 19 units of tuple characters directly: 12 can be learned from 6-item package P_2 (2 characters per tuple) and 8 from 4-item package P_3, minus 1 character that is double counted because of the 1-item overlapping. In contrast, selecting the two non-overlapping packages P_1 and P_4 only gets 12 characters.
We now define the informative score $OS_m(\cdot)$.



Package	Subtopic
P_1	$(a_1,*,c_1)$
P_2	$(*,b_1,c_1)$
P_3	$(a_2,*,c_1)$
P_4	$(*,b_1,c_2)$

Fig. 3. Summary with 4 packages. Each node is a tuple, the lines gather tuples into packages. The character $cha(t,l)$ is set to 1 for simplicity.

Definition 4 (Character Coverage). *Given a package set* $\mathcal{P}=\cup_{i=1}^{n} P_{st_i}$*, the informative score* $OS_m(\mathcal{P})$ *is defined as the Character Coverage, where the Character Coverage is the sum of all distinct characters.*

$$OS_m(\mathcal{P})=\sum_{i=1}^{n}\sum_{\substack{l\in st_i\\ t\in P_{st_i}}} cha(t,l)-\sum_{i\neq j}\sum_{\substack{l\in st_i\cap st_j\\ t\in P_{st_i}\cap P_{st_j}}} cha(t,l)+\sum_{i\neq j\neq k}\sum_{\substack{l\in st_i\cap st_j\cap st_k\\ t\in P_{st_i}\cap P_{st_j}\cap P_{st_k}}} cha(t,l)$$
$$-\cdots\cdots\pm\sum_{\substack{l\in st_1\cap\cdots\cap st_n\\ t\in P_{st_1}\cap\cdots\cap P_{st_n}}} cha(t,l) \tag{7}$$

This is an adaptation of the Inclusion-Exclusion Principle (a technique to compute the cardinality of the union of sets). $OS_m(\mathcal{P})$ has following properties.

Corollary 2. *Given a package set* $\mathcal{P}$ *and package* P_{st}*, if* $\mathcal{P}\cap P_{st}=\varnothing$*, then* $OS_m(\mathcal{P}\cup P_{st})=OS_m(\mathcal{P})+OS(P_{st})$.

Corollary 3. *Given a package set* $\mathcal{P}$ *and package* P_{st}*,* $OS_m(\mathcal{P}\cup P_{st})\leq OS_m(\mathcal{P})+OS(P_{st})$.

Corollary 2 and 3 are extended from the Inclusion-Exclusion Principle.

The TP problem now aims to find k packages with maximal character coverage $OS_m(\mathcal{P})$. This differs from previous ranking methods, since $OS_m(\mathcal{P})$ considers labels. We refer to `ComputeCCov` as the function to compute $OS_m(\mathcal{P})$.

4.2 Greedy Summarization Algorithm

Unfortunately, the objective function $OS_m(\cdot)$ is NP-hard to optimize.

Theorem 1. *The TP problem is NP-hard.*

Proof. *The basic idea is by reduction from the Weighted Maximum Coverage problem [9], which can be stated as follows. Given an integer k, and m sets $S{=}\cup_{i=1}^{m} S_i$ over a set of elements E, each element e_i is assigned with a weight $w(e_i)$. The goal is to find a k-set cover C ($C{\subseteq}S$,$|C|{=}k$) with maximum weight $\sum_{e_i \in C} w(e_i)$. The TP problems is, given a set of packages $\mathcal{P}{=}\cup_{i=1}^{n} P_{st_i}$ over a set of tuples R, each package is assigned with an overall scoring $OS(P_{st}){\propto}\sum_{t,l} cha(t,l)$. The goal is to find the set of k packages I_C ($I_C{\subseteq}\mathcal{P}$,$|I_C|{=}k$) with maximized $OS_m(I_C)$. Now, we transform an instance of the Weighted Maximum Coverage problem to an instance of the TP problem.*

Assume the attribute space size is d, and we construct d elements for each tuple t_i, denoted as $\{e_{i1},...,e_{id}\}$. There are $d|R|$ elements. Each e_{ij} is assigned with weight $w(e_{ij}){=}cha(i,j)$, where $cha(i,j)$ is the character of label j for tuple t_i. For a package P_{st_r}, we construct a set $S_r{=}\{e_{ij}|t_i{\in}P_{st_r}$,$j$ is a label in $st_r\}$. There are $|\mathcal{P}|$ sets in total. This transformation takes polynomial time. By Definition 4, $OS_m(I_C)$ calculates the sum of distinct characters, which is exactly $\sum_{e_{ij} \in C} w(e_{ij})$. It is now obvious that C maximizing $\sum_{e_{ij} \in C} w(e_{ij})$ `iff.` *the set of k packages I_C maximizes $OS_m(I_C)$.* ■

Algorithm 2 (`BS` for short) is a greedy summary algorithm. It starts by putting the package with the largest overall score into I (Line 1). At each iteration, it selects the package P_{st_i} that, together with the previously chosen packages I, produces the highest character coverage (Line 4). The algorithm stops after k packages have been chosen, and outputs I. Consider again the example in Fig. 3, when $k = 2$, `BS` produces $\{P_2,P_3\}$; and when $k = 3$, it produces $\{P_1,P_2,P_3\}$.

Algorithm 2. `BaselineSummarization`

Input: $\mathcal{P}{=}\cup_{i=1}^{n} P_{st_i}$ and k, the desired number of packages
Output: I
1 Initialize $I{=}\{\}$, and let package t be the largest overall score package in $\mathcal{P}$;
2 $I{=}I{\cup}\{t\}$, and remove t from $\mathcal{P}$;
3 **while** $|I|{<}k$ **do**
4 $\quad t{=}\max_{t \in \mathcal{P}}$(`ComputeCCov`$(I{\cup}\{t\}))$;
5 $\quad I{=}I{\cup}\{t\}$, remove t from $\mathcal{P}$;

`BS` is directly adapted from the greedy algorithm designed for *Maximum k-Set Cover problem*. It is known to have a $(1{-}1/e)$ approximation ratio [9].

`BS` computes the coverage in each iteration (Line 4), thus it can be expensive in practice. Function `ComputeCCov` has an exponential complexity, since each subpart in Eq. 7 may require the summation of an exponential number of packages. As a result, summarization by maximizing $OS_m(\mathcal{P})$ turns to be hard.

4.3 Improved Summarization Algorithm

We now present pruning conditions to reduce the invocations of `ComputeCCov`. The key observation is that, there is no intersecting tuple between packages from one meta-topic. For example, in Fig. 3, if P_1 and P_3 are from meta-topic $(Brand,{*},CPU)$, then $P_1 \cap P_3 = \varnothing$. This is obvious because each laptop has one brand and one CPU type, hence it belongs to only one package.

Formally, given a package set $\mathcal{P}$ with m meta-topics, $\mathcal{P}=\cup_{i=1}^{m} M^i$, let M^i be packages from the i-th meta-topic (i.e., $M^i=\cup_{x=1}^{|M^i|} P_{st_x^i}$, where each subtopic st_x^i belongs to the i-th meta-topic), we have $P_{st_x^i} \cap P_{st_y^i} = \varnothing$ for $x \neq y$. As described in Section 3.2, M^i is sorted in descending order of $OS(P_{st_x^i})$. Note that this package disjoint feature offers interesting information in each iteration of `BS`. We do not need to check all remaining packages to find a maximum character coverage. To give a better illustration, we first state the 3 filters and then show an example.

Specifically, for $\mathcal{P}=\cup_{i=1}^{m} M^i$, if $I=\cup_{i=1}^{m} I^i$ is the summary set in each iteration in Algorithm `BS`, $I^i=\{I_1^i, I_2^i, ...\}$ consists of packages selected from M^i, then:

1. `InitialFilter` *Assume the initial largest package t in $\mathcal{P}$ comes from M^i, if $OS(P_{st_2^i}) > OS(P_{st_1^r})$ holds for every $P_{st_1^r}$ ($r \in [1,m]$, $r \neq i$), then $P_{st_2^i}$ should be selected. This can be continued until there exists a package $P_{st_1^r}$ such that $OS(P_{st_j^i}) < OS(P_{st_1^r})$ holds for some $r \neq i$. Moreover, at the iteration that initial filter fails, only packages in M^r with $OS(P_{st_j^i}) < OS(P_{st_1^r})$ need to be checked, others can be skipped.*

2. `InclusiveFilter` *In the r-th iteration, for packages in M^i, starting from $P_{st_1^i}$, if $|I - \bigcup_{I_j^i \in I^i} I_j^i| = x$, then packages after the $(x+1)$-th package can be skipped.*

3. `ExclusiveFilter` *In the r-th iteration, for packages in M^i, starting from $P_{st_1^i}$, if $OS_m(P_{st_1^i} \cup I) - OS_m(I) = d$, then $P_{st_j^i}$ with $OS(P_{st_j^i}) \leq d$ can be skipped.*

We omit the pseudo codes of the filters and refer to them as `InitialFilter`, `InclusiveFilter` and `ExclusiveFilter`. Consider the example in Fig. 4, assume $M^1 = \{a_1, a_2, ...\}$, $M^2 = \{b_1, b_2, ...\}$, $mt_1 = (A, B, *)$ and $mt_2 = (*, B, C)$. The intersection of every two packages in M^1(or M^2) is empty, i.e. $a_i \cap a_j = \varnothing (i \neq j)$. For simplicity, we use the term "*hit*" to denote the selection of a package in each iteration, and refer to $\delta(a_i)$ as the contribution of package a_i, $\delta(a_i) = OS_m(I \cup \{a_i\}) - OS_m(I)$, thus to find a maximal character coverage can be restated as to find a package $P_{st_j^i}$ with maximal contribution $\delta(P_{st_j^i})$.

Initial Filter. The initial filter works at the beginning of `BS`. Consider the first selected package t with the largest overall score. Assume $t \in M^i$, then t is the first package in M^i (packages are sorted, i.e., $t = P_{st_1^i}$). In this case, if $OS(P_{st_2^i})$ is larger than all other $OS(P_{st_1^k})(i \neq k)$, we should select $P_{st_2^i}$ directly, for that the contribution $\delta(P_{st_2^i})$ is the largest.

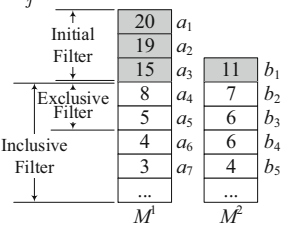


Fig. 4. Illustration of filters. Packages a_i and b_i are from meta-topics M^1 and M^2, the number in each package is its overall score.

In Fig. 4, a_1 is the largest package, thus it is firstly picked. In the 2nd iteration, a_2 hits summary set I since $OS(a_2)=19 > OS(b_1)=11$. Similarly, $|a_3|$ hits I in the 3rd iteration. With initial filter, we only need to check one package in each iteration.

In the 4th iteration, `InitialFilter` fails since $OS(a_4) < OS(b_1)$, thus we need to check all packages in M^1 and M^2 to find the next *hit*. If there is another meta-topic mt_3 with $M^3=\{c_1, c_2, ...\}$, and $OS(c_1)=5$, then we can skip M^3 since $OS(a_4) > OS(c_1)$, the contribution of all packages in M^3 is less than $\delta(a_4)=8$.

Inclusive Filter. The initial filter answers the question, *"which meta-topic should be checked to find the hitting package?"* However, when it fails, we need

to check all packages. Here, we ask another question, *"How many packages do we need to check in each meta-topic?"* Answering this leads us to inclusive filter.

Lemma 1. *Given* $M^i=\cup_{j=1}^{|M^i|}P_{st_j^i}$ *in descending order of* $OS(\cdot)$*, if* $P_{st_j^i}\cap I=\varnothing$*, then for* $k\in(j,|M^i|]$*, we have* $\delta(P_{st_k^i})\leq\delta(P_{st_j^i})$.

Proof. By Corollary 2, 3, $\delta(P_{st_k^i})=OS_m(I\cup P_{st_k^i})-OS_m(I)\leq OS(P_{st_k^i})\leq OS(P_{st_j^i})=\delta(P_{st_j^i})$. ■

By Lemma 1, we can skip $P_{st_k^i}$ $(k\in(j,|M^i|])$ if $P_{st_j^i}\cap I=\varnothing$. In Fig. 4, assume the gray packages have been selected (i.e., $I=I^1\cup I^2=\{a_1\cup a_2\cup a_3\}\cup\{b_1\}$), and it is the 5th iteration. Assume $|I^2-I^1|=8$, thus for the remaining packages $\{a_4,a_5,...\}$ in M^1, if there exists a package that has intersections with I, it can intersect 8 tuples at most, because it is disjoint with I^1. Therefore, we claim that in the next 8+1=9 ordinal packages in M^1, there must exist one package a_k $(4\leq k<4+9)$ such that $a_k\cap I=\varnothing$ (Pigeonhole principle), and by Lemma 1, packages $\{a_{k+1},a_{k+2},...\}$ can be skipped. The scale for package checking is bounded into $|I-\bigcup_j I_j^i|+1$.

Exclusive Filter. Exclusive filter calculates the overall score bound instead of scale bound. It works within each meta-topic. Consider the 5th iteration in Fig. 4, for a_4, if the contribution $\delta(a_4)=5$, then we only need to check packages with its overall score larger than 5, because the rest packages cannot hit I with a less-than-5 overall score.

Algorithm 3. `ImprovedSummarization` (`IS` for short)

Input : $\mathcal{P}=\cup_{i=1}^{m}M^i$, k is size of summary set
Output: I
1 $upperBound=\{b_1,b_2,...,b_m\}$;
2 $I=\cup_{i=1}^{m}I^i$, and initialize each I^i as $\varnothing$;
3 let package $t\in M^i$ be the largest package in $\mathcal{P}$;
4 $I^i=I^i\cup\{t\}$, remove t from $\mathcal{P}$;
5 V=`InitialFilter`$(t,M^i,\mathcal{P})$; ▷ V records the necessity of checking M^i
6 **while** $|I|<k$ **do**
7 $upperBound$=`UpdateBound`(I,V);
8 **forall the** $M^i\in\mathcal{P}$ **do** $t=\max_{t\in\{P_{st_1^i},...,P_{st_{b_i}^i}\}}$`{ComputeCCov}`$(I\cup\{t\})\}$ $I^i=I^i\cup\{t\}$, remove t from $\mathcal{P}$;

`Function UpdateBound`(I,V)
Input : $I=\{I^1,I^2,...,I^m\}$, and $V=[V_1,V_2,...,V_m]$ is the indicator for filtering
Output: $upperBound$
1 uBd_1=`InclusiveFilter`$(I,\mathcal{P},V)$, uBd_2=`ExclusiveFilter`$(I,\mathcal{P},V)$;
2 $upperBound=\{b_1,b_2,...,b_m\}$;
3 **forall the** b_i *in* $upperBound$ **do** $b_i=\min\{uBd_1[i],uBd_2[i]\}$; ▷ Choose a tighter bound

Improved Summarization Algorithm. `InclusiveFilter` and `ExclusiveFilter` both provide an upper-bound for package checking. When integrating them, the system can always choose a tighter bound to speed up the selection (see Function `UpdateBound` in Algorithm 3). Algorithm 3 summarizes packages with filters. In the first few selections, initial filter performs reduction by comparing the first packages in each meta-topic (Line 5). When `InitialFilter` fails, a boolean vector V is utilized to indicate whether each meta-topic requires checking (according to the last part of initial filter). In the following iterations, `InclusiveFilter` and `ExclusiveFilter` updates the upper-bound

by `UpdateBound` (Line 7). Algorithm 3 skips many packages, hence is faster than `BS`.

Theorem 2. *Algorithm 3 and* `BS` *produce the same* I_C. ■

The proof is omitted due to the space constraint.

5 Experimental Study

This section reports evaluations on (1) the efficiency of package generation; (2) the efficiency of summarization; and (3) the quality of summarization.

Setup. We conducted all experiments on a Windows 2008 server, with a 2.83 GHz CPU, 8 GB memory, and 1TB hard disk. The program was coded in C++.

Datasets. We used two real datasets. The **first** is a laptop dataset. It contains 511 laptops with 11 attributes, such as `Brand`, `CPU`, `Memory`, etc. We assume the whole laptop dataset as a sample query result, and denote the query as QL. The **second** is the IMDB[1] dataset. We downloaded the raw IMDB data, and preprocessed it by removing duplicate movies and missing values. A subset of the raw data was converted into a large relational table. It has 14 attributes, e.g., `Year`, `Country`, `Producer`, `Genres`. Some attributes (e.g., "actor" and "actress") may have more than one values, following [22], we picked the most frequent value if multiple values exist. After preprocessing, we have 649,506 tuples.

Our test set for IMDB data consists of 8 queries (denoted by QI_1 to QI_8). Table 5 lists the queries and the query result size. The result size is 1,236.625 on average. Note that the answers vary from different size, thus we can examine the effect when the number of tuples increases.

Table 5. Queries on IMDB dataset

QID	*Query*	$\|R\|$	QID	*Query*	$\|R\|$
QI_1	family, Christmas	366	QI_2	Revenge	507
QI_3	Legend, USA	577	QI_4	USA, Hero	1,009
QI_5	Magic	1,012	QI_6	Hong Kong Comedy	1,197
QI_7	Christmas	1,252	QI_8	short family Comedy	3,973

5.1 Package Generation

We test the efficiency of `BaselineGeneration` (`BG`) and `FG` (see Section 3.2) on three factors: (1) query result size $|R|$; (2) *threshold* (i.e., *min_sup* or *max_sup*); and (3) *attribute space size* $|\mathcal{D}|$. The support *min_sup* (*max_sup*) is proportional to package size, thus we use package size to denote *max_sup* and *min_sup*.

Fig. 5 shows the time cost for each query. Not surprisingly, `FG` outperforms `BG`, especially when $|R|$ gets larger (e.g., QI_7, QI_8). This is because `FG` could avoid producing the too small packages. The number of them gets larger when $|R|$ increases, hence the difference of time cost between `FG` and `BG` enlarges.

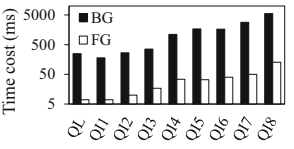
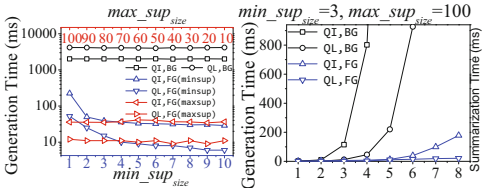


Fig. 5. Time cost on varying $|R|$. We set $|\mathcal{D}|$=5, min_sup_{size}=2, and max_sup_{size}=100.

Fig. 6a gives the time cost on varying thresholds. We set $|\mathcal{D}|$=5 and use QI to denote the average time cost of QI_1 to QI_8. The red (blue) axis shows the time cost on varying max_sup_{size} (min_sup_{size}).

[1] `ftp://ftp.fu-berlin.de/pub/misc/movies/database/`

The time cost of BG is almost unchanged, because BG generates all packages first and then removes the unacceptable ones, thus the generation time remains the same. However, the time cost of FG decreases as min_sup_{size} gets larger (see the blue lines). This is because FG does not generate the too-small-packages, and a larger min_sup_{size} often implies more too-small-packages. For max_sup_{size} (see the red lines), FG behaves the same as BG, the time cost of FG is stable. This is because when removing invalid packages, the too big ones are not removed physically (Line 4 of FG), thus the search space is not reduced.



(a) On varying sup **(b)** On varying $|\mathcal{D}|$

Fig. 6. Package Generation Performance

(a) On varying k **(b)** On varying $|\mathcal{D}|$

Fig. 8. Summarization Performance

Fig. 6b shows the performance on varying $|\mathcal{D}|$. FG outperforms BG significantly especially for a larger $|\mathcal{D}|$. As we can see, BG fails to produce acceptable packages within a reasonable amount of time (1 second) as soon as $|\mathcal{D}|$ reaches 5 or 6. This is because more attributes often implies more packages, thus FG could avoid generating more unacceptable ones.

We can conclude that FG is sufficient for real time response in most cases. This is critical in our goal of presentation. In following experiments, we set $|\mathcal{D}|$= 6, min_sup_{size}=3 and max_sup_{size}=100 by default.

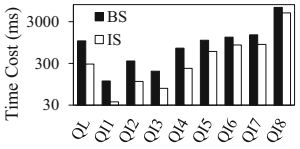


Fig. 7. Time cost on varying $|R|$

5.2 Efficiency of Summarization

We test the efficiency of Algorithm BS and IS on three factors: (1) result size $|R|$; (2) attribute space $|\mathcal{D}|$; and (3) the desired number of packages k.

Fig. 7 shows the summarization performance on varying $|R|$. The time cost is in log scale. We can see that IS outperforms BS, the time cost is reduced by 72.2% at most and 53.8% on average. Fig. 8a shows the time cost on varying the number of desired packages k. As we can see, the summarization time has been reduced 48.1% or more especially when k is large. The advantage of IS lies in the fact that it reduces the number of packages to be checked, thus the invocations of ComputeCCov are reduced. A larger $|R|$ (or k) often implies less packages for checking (compared to BS); hence the time cost is reduced greatly.

Fig. 8b shows the time cost on varying $|\mathcal{D}|$. We set $k = 5$ to reduce the advantage caused by larger k. As expected, IS still outperforms BS, especially for a larger $|\mathcal{D}|$. This is because more attributes often implies more acceptable packages, thus the number of skipping packages gets larger.

Section 5.1 and 5.2 show the efficiency of algorithms FG and IS. In this work, we analyze the properties of acceptable packages and meta-topics to perform TP

Table 6. # of acceptable packages

max_sup	10	20	**24**	50	100
QL	795	910	**928**	961	974
max_sup	10	20	**35**	50	100
QI_8	1,770	2,073	**2,208**	2,280	2,363

Table 7. Comparison of each method for QL

	# tuples	# labels	$OS_m(\cdot)$		# tuples	# labels	$OS_m(\cdot)$
BS	91	15	255	IS	91	15	255
MFR	106	10	226	SFR	98	5	105
MSC	115	11	252	SSC	99	5	105
MHS	113	10	227	SHS	64	5	105
MDS	101	15	237	SDS	98	5	105

task. Moreover, when computing, we remove (or skip) the unacceptable packages as soon as possible. Therefore, the time cost of FG and IS are reduced greatly.

5.3 Quality of Summarization

This section first validates the necessity of summarization, and then tests the quality by a case and four metrics. The character $cha(t,l)$ is set to 1.

Comparison Methods. We compared TP with several facet-ranking methods.

(1)FR Frequency-based ranking [6], where facets are ranked by the number of tuples in them (i.e., the larger support a facet has, the higher it ranks).

(2)SC Set-cover ranking [6], where k facets are selected to maximize the union of tuples in these facets.

(3)HS Hill-climbing Selection [12], where k facets are selected by hill climbing technique. We define the cost as the number of tuples exposed to users.

(4)DS Deterministic Selection, where a set of top-k largest overall scoring packages are chosen (an adaptation of frequency-based ranking with $OS(\cdot)$).

For faceting, there are single facet and multi facets. In single facet, the tuples are partitioned according to 1 attribute, and in multi-facets, m attributes. We compare with both of them. In total, we got 7 competitors: 4 ranking methods, each of them with 2 kinds of faceting. Note that for single facet, DS and FR are the same since $cha(t,l) = 1$. We denote the 7 competitors as SFR (equivalent to SDS), SSC, SHS, MFR, MSC, MHS, MDS. For a code 'XYZ', X indicates the size of facets, i.e., S for single facet, M for multi-facets. YZ indicates the ranking method. In following experiments, we also use YZ to denote both SYZ and MYZ.

Necessity. Table 6 describes the number of acceptable packages grows with max_sup_{size}. The support value 24 and 35 are estimated by Eq. 6. Note that even when max_sup_{size}=10, the number of acceptable packages reaches into hundreds or thousands, it is too large for a user. This result clearly shows that obtaining a summary of packages is necessary for presentation.

Case study. Table 7 gives a case for QL, and Fig. 9 shows the subtopics returned by 5 methods. In this case, we chose 5 attributes: Brand, CPU, Memory, HardDrive, GraphicsCard, and set min_sup_{size}=3, max_sup_{size}=24, and k=5.

As we can see, IS returns more labels with a larger character coverage. Fig. 9a shows that labels returned by IS are more diverse than others, it contains labels from all 5 attributes. Moreover, if we measure the diversity of labels by averaging the number of unique labels in each attribute (i.e., $LabelDiv(I_C) = \Sigma_{A_i \in \mathcal{D}} \text{Number of unique labels in } A_i / |\mathcal{D}|$), we can find that IS has the largest $LabelDiv(I_C)$. The **label diversity** for IS, MDS, MSC, MFR and MHS are 2, 1.8, 1.6, 1.8 and 1.8, respectively. Therefore IS tends to produce informative packages. However, IS returns less tuples in Table 7. We compare these methods in detail next.

Acer	4 GB	500 GB	22
HP	4 GB	Intel Graphics	19
Lenovo	4 GB	500 GB	17
Intel i7	8 GB	750 GB	16
4 GB	640 GB	Intel Graphics	20

(a) IS

Acer	4 GB	500 GB	22
4 GB	640 GB	Intel Graphics	20
HP	4 GB	320 GB	20
Lenovo	4 GB	320 GB	20
Dell	320 GB	Intel Graphics	19

(b) MDS

640 GB	Intel Graphics		24
Lenovo	500 GB		24
Dell	320 GB		23
Lenovo	320 GB		22
Acer	4 GB	500 GB	22

(c) MSC

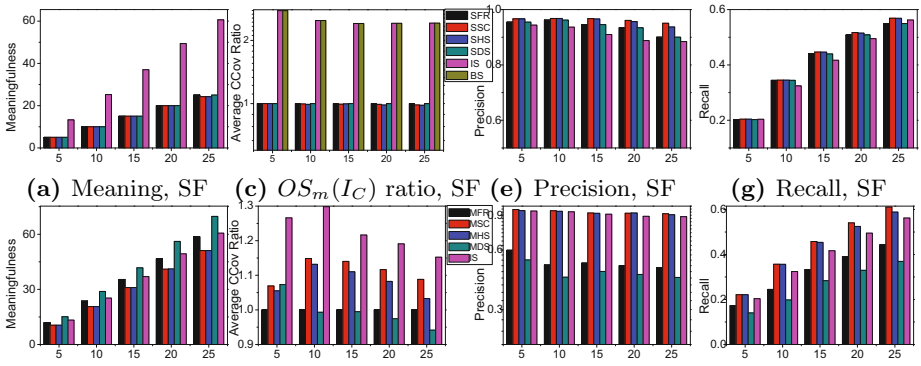
640 GB	Intel Graphics		24
Intel i3	4 GB		24
Lenovo	500 GB		24
Dell	320 GB		23
6 GB	500 GB		22

(d) MFR

Toshiba	6 GB		22
Intel i3	4 GB		24
Dell	320 GB		23
4 GB	750 GB		21
Lenovo	500 GB		24

(e) MHS

Fig. 9. A Case on QL, each facet is followed by the number of tuples in it



(a) Meaning, SF **(c)** $OS_m(I_C)$ ratio, SF **(e)** Precision, SF **(g)** Recall, SF

(b) Meaning, MF **(d)** $OS_m(I_C)$ ratio, MF **(f)** Precision, MF **(h)** Recall, MF

Fig. 10. Summarization quality on varying k. SF stands for 1-facet, and MF multi-facets

Metrics. For TP and its 7 competitors, we compare (1) meaningfulness, (2) character coverage, (3) precision and (4) recall, where,

$$precision(I_C)=\frac{\#\text{ distinct tuples in } I_C}{\#\text{ tuples presented to users}},\ recall(I_C)=\frac{\#\text{ distinct tuples in } I_C}{|R|} \quad (8)$$

1.Meaningfulness Fig. 10a and 10b compare the meaningfulness. As we can see, IS and MDS have more meanings than FR, SC and HS. Recall that FR returns the top-k support facets, but such facets may often be less meaningful (e.g., in Fig. 9, MFR only returns 10 labels). On the other hand, the subtopics returned by IS are as different as possible from each other so as to get a higher character coverage, thus are likely to be meaningful (e.g., IS gets 15 labels for QL).

The SC selects facets by maximizing the union of tuples, thus it tends to return facets with more tuples and less overlapping (e.g., the facets of MSC have more tuples than IS and MDS in Fig. 9). Due to the inverse proportion of package size and meaningfulness, the meaning of SC is limited (e.g., MSC gets 11 labels). The HS finds a local optimal coverage for tuples, thus the meaning is also limited.

Note that MDS has more meaning than IS; this is because MDS selects packages with the highest overall scores, and the meaningfulness is a factor to be considered. However, there are significant overlapping between facets, hence the meanings are also overlapped (e.g., MDS has less label diversity than IS for QL).

2.Character coverage Fig. 10c and 10d show that BS and IS produce the highest character coverage. Note that IS achieves exactly the same character coverage as BS, which confirms the correctness of our filters in Section 4.3. Therefore, in following evaluations we compare only IS with other methods.

FR and DS return facets with largest support or overall scoring, which may neglect the overlapping between facets (e.g., the label diversity of FR and DS is less than IS for QL). Hence, they fail to return facets with high character coverage. On the other hand, MSC and MHS have relatively higher character coverage. This is because SC and HS aim to maximize the union of tuples, thus are likely to return disjoint facts (e.g., Lenovo 500GB and Lenovo 320GB in Fig. 9c). Therefore, they tend to have more distinct characters.

3.Precision Fig. 10e and 10f show the precision of each method. As we can see, SC and HS get higher score, because they are likely to have less overlaps than IS, FR and DS due to their optimization goal. For FR and DS, the overlapping is significant especially for MFR and MDS (e.g., in Fig. 9, MFR has 11 overlapped tuples, whereas IS has 3). Therefore, their precisions are low. On the other hand, the precision of IS is close to others especially when k is small (e.g., $k < 15$), and outperforms MFR and MDS significantly. This is because when k gets larger, the overlap between facets may increase (IS aims to find maximal character coverage instead of maximal tuple coverage). However, in practice, the presentation task prefers good quality with a relatively small k to avoid overwhelming.

4.Recall Fig. 10g and 10h compare the recall. We can see that IS is close to SC and HS, also it outperforms FR and DS on most cases. SC and HS are designed to get the largest tuple coverage, thus they have strong overview ability (e.g., the distinct tuples of MSC and MHS ranks the top 2 largest in Table 7). We can conclude that IS has slightly more overlapped tuples than SC and HS, hence its overview ability is close to them.

The four metrics test the quality differently. *Meaningfulness* evaluates the meaning of packages; *character coverage* tests both tuples and labels collectively; *precision* evaluates the overlaps and *recall* the overview ability. As we can see, most existing methods aim to optimize precision and recall, rather than the meaning. In this work, IS combines meaningfulness and overview ability by character coverage. The experiments show that IS improves the meaning by 49.1% on average, and losses overview ability slightly by 6.2% at most. Given the superior performance, we can conclude that IS produces promising summaries.

6 Related Work

Facets. Faceted search has been extensively studied in DB community [4,16,6,12]. Recently, facet-ranking methods are proposed for the facet overwhelming problem [6,12]. However, most of them focus the number of tuples in each facet; they neglect to rank facets with labels. Besides, [3] proposes a probabilistic ranking model, whereas it is designed for documents. On the other hand, [4,6,16] rank facets by a cost model to minimize the user efforts. They are different from TP, since TP does not involve the interaction of users. Bin [22] answers aggregate queries, which is similar to subtopics in this paper. However, no further summarizing was performed for the overwhelming problem.

Clustering. Clustering helps users search the answer space (see [2] for a survey). Various methods [20,13,17] present the results by finding the naturally close

groups in answer space. However, the clusters are not intelligible. [17] proposes a labelling clustering method for web pages, whereas it is hard to adopt on relational data for real time response. [13] is another effective clustering method, it reduces the tuple overwhelming by further clustering results in each cluster.

Ranking. Ranking and top-k query answering [1,5,14,15] rank results by structural or statistical information. It is effective, but for navigational queries, users need to navigate the answer space, hence faceting and clustering are proposed.

Diversification. Diversification could provide more different results. In DB community, some pioneering approaches [13,7,21,10] have emerged. Bin [13] diversifies results by choosing representatives from each cluster. DivQ [7] aims to discover diversified schemas. BROAD [21] captures both structure and semantic information by a kernel distance metric. These approaches provide different results, whereas TP provides packages with maximal information.

User Interface. User interfaces (e.g., Skimmer[18], MusiqLens[13], and DataScope[19]) are designed to give a fast overview of answer space, and meanwhile feed users a small fraction of answers. Clustering or sampling is performed.

7 Conclusion

In this paper, we had proposed TP for presenting search results on database. To our best knowledge, TP is the first work that addresses the presentation from both package size and labels. First, we identified acceptable packages and designed a fast algorithm to generate them. Then we quantified the character coverage and proposed algorithms for summarization. The experimental results show that TP yields promising summaries efficiently. In future, we would like to integrate TP with existing ranking methods.

References

1. Agrawal, S., Chaudhuri, S., Das, G., Gionis, A.: Automated ranking of database query results. In: CIDR (2003)
2. Carpineto, C., Osinski, S., Romano, G., Weiss, D.: A survey of web clustering engines. ACM Comput. Surv. 41(3) (2009)
3. Carterette, B., Chandar, P.: Probabilistic models of ranking novel documents for faceted topic retrieval. In: CIKM, pp. 1287–1296 (2009)
4. Chakrabarti, K., Chaudhuri, S., won Hwang, S.: Automatic categorization of query results. In: SIGMOD, pp. 755–766 (2004)
5. Chaudhuri, S., Das, G., Hristidis, V., Weikum, G.: Probabilistic information retrieval approach for ranking of database query results. ACM TODS 31(3), 1134–1168 (2006)
6. Dakka, W., Ipeirotis, P.G., Wood, K.R.: Automatic construction of multifaceted browsing interfaces. In: CIKM, pp. 768–775 (2005)
7. Demidova, E., Fankhauser, P., Zhou, X., Nejdl, W.: DivQ: Diversification for keyword search over structured databases. In: SIGIR, pp. 331–338 (2010)
8. Donohue, J.C.: Understanding scientific literatures: A Bibliometric Approach. The MIT Press, Cambridge (1973)

9. Hochbaum, D.S. (ed.): Approximation algorithms for NP-hard problems. PWS Publishing Co., Boston (1997)
10. Hu, H., Zhang, M., He, Z., Wang, P., Wang, W.: Diversifying query suggestions by using topics from wikipedia. In: Web Intelligence, pp. 139–146 (2013)
11. Koller, D., Sahami, M.: Hierarchically classifying documents using very few words. In: ICML, pp. 170–178 (1997)
12. Li, C., Yan, N., Roy, S.B., Lisham, L., Das, G.: Facetedpedia: Dynamic generation of query-dependent faceted interfaces for wikipedia. In: WWW, pp. 651–660 (2010)
13. Liu, B., Jagadish, H.V.: Using trees to depict a forest. PVLDB 2(1), 133–144 (2009)
14. Luo, Y., Lin, X., Wang, W., Zhou, X.: Spark: Top-k keyword query in relational databases. In: SIGMOD, pp. 115–126 (2007)
15. Luo, Y., Wang, W., Lin, X., Zhou, X., Wang, J., Li, K.: Spark2: Top-k keyword query in relational databases. IEEE Trans. Knowl. Data Eng. 23(12), 1763–1780 (2011)
16. Roy, S.B., Wang, H., Das, G., Nambiar, U., Mohania, M.K.: Minimum-effort driven dynamic faceted search in structured databases. In: CIKM, pp. 13–22 (2008)
17. Scaiella, U., Ferragina, P., Marino, A., Ciaramita, M.: Topical clustering of search results. In: WSDM, pp. 223–232 (2012)
18. Singh, M., Nandi, A., Jagadish, H.V.: Skimmer: Rapid scrolling of relational query results. In: SIGMOD Conference, pp. 181–192 (2012)
19. Wu, T., Li, X., Xin, D., Han, J., Lee, J., Redder, R.: Datascope: Viewing database contents in google maps' way. In: VLDB, pp. 1314–1317 (2007)
20. Zeng, H.-J., He, Q.-C., Chen, Z., Ma, W.-Y., Ma, J.: Learning to cluster web search results. In: SIGIR, pp. 210–217 (2004)
21. Zhao, F., Zhang, X., Tung, A.K.H., Chen, G.: Broad: Diversified keyword search in databases. PVLDB 4(12), 1355–1358 (2011)
22. Zhou, B., Pei, J.: Answering aggregate keyword queries on relational databases using minimal group-bys. In: EDBT, pp. 108–119 (2009)