

# Subspace Discovery for Promotion: A Cell Clustering Approach\*

Tianyi Wu and Jiawei Han

University of Illinois at Urbana-Champaign, USA  
{twu5,hanj}@illinois.edu

**Abstract.** The promotion analysis problem has been proposed in [16], where ranking-based promotion query processing techniques are studied to effectively and efficiently promote a given object, such as a product, by exploring ranked answers. To be more specific, in a multidimensional data set, our goal is to discover interesting subspaces in which the object is ranked high. In this paper, we extend the previously proposed promotion cube techniques and develop a cell clustering approach that is able to further achieve better tradeoff between offline materialization and online query processing. We formally formulate our problem and present a solution to it. Our empirical evaluation on both synthetic and real data sets show that the proposed technique can greatly speedup query processing with respect to baseline implementations.

## 1 Introduction

The *promotion analysis problem* [16] aims to search for interesting subspaces for some user-specified target object such as a person or a product item so that it can be promoted in the subspaces discovered. Such a function is called *ranking-based promotion query*. Given a user-specified object, a promotion query should return the most interesting subspaces in the multidimensional lattice, where “interesting” intuitively means that the object is among the top ranked answers in a particular subspace. It has been shown that many OLAP and decision support applications can potentially benefit from such a function, as data analysts may find it useful to explore the search space to promote a target object. For example, suppose a car model is given as the target object, then interesting locations may be subsequently discovered such that the car model is highly ranked in terms of sales or customer review in those locations, which brings up opportunities for promotion.

To process promotion queries efficiently, however, is a challenging problem. In this paper, we focus on the offline side of the problem. We extend the previously studied promotion cube technique [16] and propose a general *cell clustering approach* to further achieve better query execution time vs. materialization

---

\* The work was supported in part by the U.S. National Science Foundation grants IIS-08-42769 and BDI- 05-15813, and the Air Force Office of Scientific Research MURI award FA9550-08-1-0265.

$A_1$	$A_2$	$O$	$M$
$a_1^1$	$a_2^2$	$o_1$	0.3
$a_1^1$	$a_2^2$	$o_3$	0.2
$a_1^1$	$a_3^2$	$o_1$	0.5
$a_1^1$	$a_3^2$	$o_3$	0.6
$a_1^1$	$a_3^2$	$o_1$	0.8
$a_2^1$	$a_3^2$	$o_2$	0.5
$a_2^1$	$a_2^2$	$o_3$	0.6
$a_2^1$	$a_2^2$	$o_2$	0.3
$a_2^1$	$a_2^2$	$o_1$	0.3

Fig. 1. Multidimensional table

tradeoff. Our observation is that promotion cells can be clustered together in a coherent representation, while at the online processing step the uninteresting subspaces can still be effectively pruned out.

**Example 1.** Before introducing our proposed techniques, let us first examine a concrete example of promotion analysis. Figure 1 shows an example multidimensional table with two dimensions  $A_1$  and  $A_2$ , each having 2 distinct values. In addition, the object dimension is represented by  $O$ , and the measure dimension is a numerical dimension represented by  $M$ . Each row in this table is called a base tuple and by enforcing selection conditions over  $A_1$  and  $A_2$  we can obtain different object subspaces. Figure 2 shows all nine subspaces obtained from the example multidimensional table. Among these subspaces, the first  $\{*,*\}$  is a special one called full space. For each of these subspaces, we show its ranked list of aggregates based on the AVG aggregation. For example, for the subspace  $\{A_1 = a_1^1, A_2 = *\}$ ,  $o_1$ 's AVG can be computed as  $(0.3 + 0.5 + 0.8)/3 = 0.53$ .

A straightforward offline method for processing the promotion query is to materialize all aggregates, e.g., store Table 2 completely (notice that the objects are shown in the table for reference, but they do not need to be stored). Now, given a target object, say  $o_3$ , one can easily locate the most interesting subspaces  $\{a_2^1, *\}$ ,  $\{*, a_2^2\}$ , and  $\{a_2^1, a_2^2\}$ , because it is ranked the very first in those subspaces. Similarly, given the aggregate measure, any other target object can be processed in the same fashion using these completely precomputed results.

Since there could be a very large number of subspaces as well as objects even for a moderately large data set, the precompute-all strategy can be very costly at the offline stage. To this end, a general promotion cube framework has been proposed as a partial precomputation strategy that avoids such high cost. In a promotion cube, each promotion cell consists of a set of summary aggregates to help bound the target object's rank. However, in most cases it is even unnecessary to store all promotion cells since object aggregates tend to be similar across different subspaces. This leads to our proposed clustering approach to further remove redundancy. After clustering, the number of materialized aggregates can be largely reduced, while the efficiency of online query execution can still be guaranteed. Another advantage is that such a clustered cube structure does not

$A_1$	$A_2$	Ranked list of aggregates
*	*	$(o_1)$ 0.48, $(o_3)$ 0.47, $(o_2)$ 0.4
$a_1^1$	*	$(o_1)$ 0.53, $(o_3)$ 0.4
$a_2^1$	*	$(o_3)$ 0.6, $(o_2)$ 0.4, $(o_1)$ 0.3
*	$a_2^2$	$(o_3)$ 0.4, $(o_1)$ 0.3, $(o_2)$ 0.3,
*	$a_3^2$	$(o_1)$ 0.65, $(o_3)$ 0.6, $(o_2)$ 0.5
$a_1^1$	$a_2^2$	$(o_1)$ 0.3, $(o_3)$ 0.2
$a_1^1$	$a_3^2$	$(o_1)$ 0.65, $(o_3)$ 0.6
$a_2^1$	$a_2^2$	$(o_3)$ 0.6, $(o_1)$ 0.3, $(o_2)$ 0.3
$a_2^1$	$a_3^2$	$(o_2)$ 0.5

**Fig. 2.** Fully precomputed results using *AVG* aggregation

restrict the capability of promotion analysis in that any promotion query can be supported for a given type of aggregation. In this paper, we

- extend the promotion cube framework to a cell clustering approach to further achieve better balance between offline materialization cost and online query execution cost;
- formally define the promotion cube structure with clustered cells;
- discuss the hardness of the problem and develop a greedy algorithm to perform clustering so that the quality of result is guaranteed; and
- conduct empirical evaluation on synthetic as well as real-world data sets to verify the performance of the cell clustering approach, and show that the approach is superior to baseline approaches.

The remainder of the paper is organized as follows. Section 2 discusses related work. Section 3 formulates the problem of the paper. In Section 4 we propose the cell clustering approach based on the promotion cube framework. Section 5 reports experimental results, and finally, Section 6 concludes this work.

## 2 Related Work

The promotion analysis problem is originally studied in [16], which proposes the promotion query model and its query processing techniques; also, a statistical method is discussed to prevent spurious promotion results. Our paper can be regarded as a follow-up study of it in that the underlying multidimensional data model as well as the promotion query model remain the same. However, the main focus of this study is different from the previous work; that is, the major goal here is to propose a new offline strategy to perform clustering over promotion cells, while being able to answer online promotion queries on multidimensional data in an efficient way.

Besides, this work is related to several previous studies on database ranking techniques. The ranking cube method is first studied in [17] for answering top- $k$  queries which allows users to enforce conditions on database attributes.

Subsequently, [15] discusses the problem of processing top- $k$  cells from multidimensional group-by's for different types of measures. Both of the above methods leverage offline precomputation to achieve better online performance. Moreover, the threshold algorithms [6] represent yet another family of ranking techniques, where the objective is to efficiently produce the top answers by aggregating scores from multiple ranked lists. Other variants of ranking algorithms have also been studied and applied in a variety of applications including Web-accessible databases [12], supporting expensive predicates [2], keyword search [10], and visualization [14]. Unfortunately, none of these techniques can handle our problem.

Various clustering problems are also related to our study. [13] presents the *RankClus* method for integrating ranking and clustering such that the results of both can be mutually enhanced. Earlier papers like [5,18] have discussed methods for compressing and/or clustering data points to facilitate further data mining. However, these methods are different from the one in this paper since our clustered cells are for pruning. In addition, our work shares similar objectives as some previous clustering algorithms. For example, [3,1,4] propose approximation algorithms to clustering data points so as to minimize a particular objective (e.g., the sum of cluster diameters). Our cell clustering problem can fit into their problem setting but their results are mainly of theoretical interests and may not scale to large data. There are also other related studies in that multidimensional analysis is conducted for data mining [11,7], but none of them can be applied toward our context.

### 3 Problem Formulation

To ensure that the discussion be self-contained, we present the formulation of the promotion query problem in this section. Consider a multidimensional table  $T$  consisting of a collection of base tuples. There are three categories of columns:  $d$  categorical dimensions,  $A_1, \dots, A_d$ , which are called *subspace dimensions*, a column  $O$  storing objects called *object dimension*, and a numerical *score dimension*,  $M$ . We use  $O$  to represent also the set of objects. Based on the multidimensional schema, we call  $S = \{a_1, a_2, \dots, a_d\}$  a *subspace*, where  $a_i$  is either a dimension value of  $A_i$  or it is “any” value denoted by “\*”. moreover, the set of all subspaces is denoted by  $\mathbf{U}$ .

Given an aggregate function  $\mathcal{M}$  (e.g., *AVG*), one can derive for each subspace the ranked list of objects and their aggregate values.

The promotion analysis problem can be formulated as follows. Given a *target object*  $\tau$  for promotion ( $\tau \in O$ ), we let  $\mathbf{V} = \{S | \tau \text{ occurs in subspace } S \wedge S \in \mathbf{U}\}$ , i.e., the set of subspaces where  $\tau$  occurs. Our goal is to *discover the top- $k$  subspaces in  $\mathbf{V}$  such that  $\tau$  is the most highly ranked in these  $k$  subspaces*, where  $k$  is a user parameter.

**Example 2.** *In the running example shown in Table 1, we can see that  $A_1$  and  $A_2$  are the subspace dimensions,  $O$  is the object dimension, and  $M$  is the score dimension. Table 2 shows  $\mathbf{U}$ , i.e., the set of all 9 subspaces. Now suppose the target object is  $\tau = o_3$  and the aggregate function is  $\mathcal{M} = \text{AVG}$ . Then  $|\mathbf{V}| = 8$*

because  $o_3$  has 8 subspaces containing  $\tau$  ( $\{a_2^1, a_3^2\}$  is not in  $\mathbf{V}$  because  $o_3$  does not appear in it). If  $k = 3$ , the top- $k$  subspaces for  $o_3$  should be  $\{a_2^1, *\}$ ,  $\{*, a_2^2\}$ , and  $\{a_2^1, a_2^2\}$  because  $o_3$  is ranked first in all of them but it has lower rank in any other subspace.

## 4 The Cell Clustering Approach

To handle the subspace discovery problem efficiently, we propose to study a cell clustering approach within the promotion cube framework. We present the cube structure in Section 4.1, followed by a brief description of the complementary online algorithm in Section 4.2. Section 4.3 discusses an optimization technique for clustering.

### 4.1 Promotion Cube with Clustered Cells

To support data analysis for large-scale applications, computing the top- $k$  subspaces from scratch may not be a good solution. On the other extreme, materializing all cells in a promotion cube may suffer from excessive space requirement due to a blow-up of the number of subspaces (which can be up to  $O(\prod_{i=1}^d |A_i|)$ , where  $A_i$  denotes the cardinality of dimensions  $A_i$ ). To tackle this problem, we observe that aggregates in different subspaces tend to be similar, so it would be wasteful to materialize all cells separately. Thus, we propose a clustering approach to reduce the space overhead of the promotion cube. This structure presents a summary of aggregated data and meanwhile is able to facilitate online exploration.

Given multidimensional table  $T$  and aggregate function  $\mathcal{M}$ , The definition of the cube structure with clustered cells is as follows.

**Definition 1.** Given any subspace  $S \in \mathbf{U}$ , a clustered cell,  $CluCell(S)$ , is defined as a collection of upper and lower bound aggregates; it specifically consists of  $2n$  values,  $(\bar{c}_1, \underline{c}_1; \bar{c}_2, \underline{c}_2; \dots; \bar{c}_n, \underline{c}_n)$ , where  $\bar{c}_i$  and  $\underline{c}_i$  ( $1 \leq i \leq n$ ) are the upper and lower bounds for the  $((i - 1) \cdot w + 1)$ -th largest object aggregate value in subspace  $S$ . Here  $n$  and  $w$  are positive integers.

The promotion cube with clustered cells, denoted by  $CluPromoCube$ , is defined as a collection of clustered cells in the format  $(S, CluCell, Sup)$  for all subspaces qualifying some  $minsup$  threshold; that is,  $CluPromoCube = \{S : CluCell(S), Sup(S) \mid Sup(S) > minsup\}$ .

Here  $Sup(S)$  refers to how many objects subspace  $S$  has. In the above definitions  $n$  and  $w$  are user-specified parameters that determine the size of each materialized cell, whereas  $minsup$  is another parameter that dictates which cells to materialize. These parameters are application-dependent. When  $minsup = 0$ ,  $n = |\mathcal{O}|$ ,  $w = 1$ , and  $\bar{c}_i = \underline{c}_i$ , the promotion cube would degenerate to fully precomputing all results since all subspaces and all object aggregate scores are precomputed. In fact we often have  $minsup > 0$  and  $n \ll |\mathcal{O}|$  (i.e., only a small fraction of aggregate values will be materialized), and  $\bar{c}_i > \underline{c}_i$  (i.e., several promotion cells will be clustered together).

<i>Subspace</i>	<i>CluCell(S)</i>	<i>Sup(S)</i>
$S_1$	(8.9,8.8; 7.0,6.8; 3.4,3.0; 1.1,0.9; 0.3,0.3)	200
$S_2$	(5.1,5.0; 3.6,3.2; 1.4,1.2; 0.8,0.5; 0.05,0.01)	150
$S_3$	(9.1,8.5; 7.2,6.1; 3.2,2.9; 1.1,1.0; 0.2,0.02)	260
$S_4$	(9.0,8.8; 6.9,6.0; 3.3,3.0; 0.9,0.5; 0.2,0.1)	220
...	...	...

**Fig. 3.** An example *CluPromoCube* with  $n = 5$ ,  $w = 10$ , and  $minsup=100$

**Example 3.** Figure 3 illustrates an example of clustered cells (not following the previous examples), where for each subspace passing  $minsup = 100$ , a clustered cell ( $n = 5$ ,  $w = 10$ ) is materialized. For instance, subspace  $S_3$  contains  $2 \times n = 10$  aggregate values, where  $\bar{c}_2 = 7.2, \underline{c}_2 = 6.1$  indicate that the  $(i - 1) \cdot w + 1 = (2 - 1) \cdot 10 + 1 = 11$ th largest aggregate is no more than 7.2 and no less than 6.1.

### 4.2 Online Algorithm

Given the offline data structure *CluPromoCube*, we briefly describe the complementary online algorithm *PromoRank* to produce top- $k$  subspaces for any input  $\tau$  [16]. Notice that other online algorithms may also work with *CluPromoCube* in a similar way but they are beyond the scope of this study. *PromoRank* first computes  $\mathbf{V}$ , the subspaces containing  $\tau$  along with  $\tau$ 's aggregate values. Second, a candidate set of subspaces is generated based on *CluPromoCube*. Third, the candidate set is aggregated to produce the correct and complete set of top- $k$  results. This algorithm is depicted in Table 1.

At the beginning,  $\mathbf{V}$  is obtained and for each  $S \in \mathbf{V}$ ,  $\tau$ 's aggregate value is computed. For these  $|\mathbf{V}|$  subspaces, we denote  $\tau$ 's aggregates as  $M_1, M_2, \dots, M_{|\mathbf{V}|}$ . To compute the aggregates, a depth-first enumeration of all  $\tau$ 's tuples would be enough. Then, a candidate subspace set is generated whereas non-candidate subspaces are pruned. The generation of the candidate set requires us to first compute  $HRank_s$  (i.e., the highest possible rank) and  $LRank_s$  (ie, the lower possible rank) for each subspace  $S_s$  ( $1 \leq s \leq |\mathbf{V}|$  and  $S_s \in \mathbf{V}$ ) based on *CluPromoCube*.

When some  $S_s$  does not meet the  $minsup$  threshold, we cannot derive  $\tau$ 's rank in it since it becomes unbounded (if the final top- $k$  subspaces are also required to satisfy the  $minsup$  condition,  $S_s$  can be pruned immediately). Otherwise, given  $S_s$ 's clustered promotion cell  $CluCell(S_s) = (\bar{c}_1, \underline{c}_1; \bar{c}_2, \underline{c}_2; \dots; \bar{c}_n, \underline{c}_n)$ ,  $HRank$  and  $LRank$  can be computed as in the following cases:

- Let  $i$  be the smallest value in  $\{1, 2, \dots, n\}$  such that  $M_s > \bar{c}_i$ , then  $LRank_s = (i - 1) \cdot w$ . If such  $i$  does not exist, we have  $LRank_s = Sup(S_s)$ ;
- Let  $j$  be the largest value in  $\{1, 2, \dots, n\}$  such that  $\underline{c}_j > M_s$ , then  $HRank_s = (j - 1) \cdot w + 2$ . If such  $j$  does not exist, we have  $HRank_s = 1$ .

Note that the above computation assumes that there is no duplicate aggregate value within any subspace. In the presence of duplicate aggregates, the computation can be easily extended. Let  $R_k$  be the  $k$ -th largest  $LRank_s$  for  $1 \leq s \leq |\mathbf{V}|$ .

**Table 1.** The online processing algorithm

```

Algorithm 1
(Aggregate  $\mathbf{V}$ )
1 Compute  $M_1, M_2, \dots, M_{|\mathbf{V}|}$ , the aggregate values of  $\tau$  in each subspace
  in  $\mathbf{V}$ ;
  (Prune out non-candidate subspaces)
2 for  $s = 1 \rightarrow \mathbf{V}$  do compute  $HRank_s$  and  $LRank_s$  using
   $CluPromoCube$ ;
3  $R_k \leftarrow$  the  $k$ -th largest value among  $\{LRank_s | 1 \leq s \leq |\mathbf{V}|\}$ ;
4 Prune out all subspaces having  $HRank_s > R_k$ ;
  (Generate the complete set of results)
5 Call the recursive procedure below on  $(\{*\}, T, 0)$  to compute  $\tau$ 's rank
  in each unpruned subspace;
6 Return the top- $k$  subspaces where  $\tau$  has the highest ranks;
  (Below is a recursive procedure on  $(S, T, d_0)$  )
7 if  $S$  is not pruned then compute  $\tau$ 's exact rank in  $S$ ;
8 for  $d' \leftarrow d_0 + 1$  to  $d$  do
9   Sort  $T$ 's tuples based on  $d'$ -th subspace dimension;
10  for each distinct value  $v \in A_{d'}$  do
11     $S' \leftarrow S \cup \{d' : v\}$ ; /* next subspace */
12     $T' \leftarrow T$ 's tuples having  $v$  on  $A_{d'}$ 
13    Recursively call  $(S', T', d')$  if  $S'$  contains  $\tau$ ;
14  end
15 end
    
```

Any subspace  $S_s$  with  $HRank_s$  lower than  $R_k$  (i.e.,  $HRank_s > R_k$ ) must not be a top- $k$  result and thus can be pruned. As a result, the unpruned subspaces form the candidate set which must be a superset of the final top- $k$  subspaces. Notice that if the exact order of the top- $k$  results is not required, one may directly output the subspaces whose  $LRank$  is greater than the  $k$ -th highest  $HRank$  without adding them into the candidate set.

**Example 4.** Given the  $CluPromoCube$  in Example 3, Figure 4 illustrates the computation of  $HRank$  and  $LRank$  for some  $\tau$ . For instance, suppose  $\tau$ 's aggregate value for subspace  $S_2$  has been computed as  $M_2 = 3.5$ . Based on the corresponding clustered cell for  $S_2$  in Figure 3, we have that  $M_2 > \bar{c}_3$ , meaning that  $i = 3$  and  $LRank_2 = (i - 1) \cdot w = 20$ ; also we have  $HRank_2 = (j - 1) \cdot w + 2 = 2$  because

Subspace	$M_s$	$HRank$	$LRank$
$S_1$	0.2	42	200
$S_2$	3.5	2	20
$S_4$	5.0	12	20
...	...	...	

**Fig. 4.** Example  $HRank$  and  $LRank$  computation based on  $CluPromoCube$

$\underline{c}_j > M_2$  holds when  $j = 1$ . Similarly for  $S_1$  and  $S_4$  we obtain their *HRank* and *LRank* respectively as shown in the figure. Thus,  $S_1$  can be pruned when  $k = 2$  because  $R_k$  would be no more than 20 while for  $S_1$ , its *HRank* is 42.

Finally, the exact ranks of  $\tau$  are evaluated for the unpruned subspaces using a recursive procedure displayed in Table 1 (Lines 7–15). It starts with the full space and recursively sorts data to generate children subspaces in a depth-first manner. The multidimensional table  $T$  is iteratively sorted according to the  $d'$ -th dimension (Line 9), such that each dimension value extends to a new child subspace  $S'$  (Line 11). For the current subspace  $S$ , if it is in the candidate set,  $\tau$ 's exact rank would be derived by aggregating objects in the input table  $T$  (Line 7). In this way, all subspaces in  $\mathbf{V}$  can be enumerated. Although the bottleneck of the online algorithm lies in the recursive procedure, where the worst-case time complexity could be  $O(|\mathbf{V}| \cdot T)$ , *CluPromoCube* is able to help prune many uninteresting subspaces, thereby bringing down the total online cost. Note that the size of *CluPromoCube* would only affect candidate generation and thus the performance.

### 4.3 Clustered Cell Generation

We now turn to the generation of clustered cell at the offline stage. Given the multidimensional table, two parameters  $n$  and  $w$ , we can generate promotion cells for all subspaces by aggregating this table and then select  $n$  aggregates at every  $w$ -th position. Doing multidimensional aggregation and generating such selected aggregates for each subspace have been well-studied [7] and thus we do not provide details here. Our focus is on how to generate clustered cells.

For each subspace, let us call its promotion cell, *i.e.*, the  $n$  selected aggregates, an  $n$ -dimensional *point* in the Euclidean space. Therefore, our problem is to cluster  $|\mathbf{U}|$  points into a few clusters. For example, given two 5-dimensional points (5.0, 4.0, 3.0, 2.0, 1.0) and (5.5, 3.6, 3.1, 1.8, 0.05), they can be clustered into a *CluCell*, which is (5.5, 5.0; 4.0, 3.6; 3.1, 3.0; 2.0, 1.8; 1.0, 0.05). In principle, a *CluCell* can be generated from multiple promotion cells by taking the maximum and minimum aggregate value at each of the  $n$  dimensions. Note that such a clustering approach does not affect the correctness and completeness of results by the definition of *CluCell*. Since clustering two distant points may adversely affect the online algorithm, we use the following distance function:

**Definition 2.** *Given two  $n$ -dimensional points in the Euclidean space,  $P_1 = (p_1^1, p_2^1, \dots, p_n^1)$  and  $P_2 = (p_1^2, p_2^2, \dots, p_n^2)$ , let their distance be the  $L_1$ -distance:  $dist(P_1, P_2) = \|P_1, P_2\|_1 = \sum_{i=1}^n |p_i^1 - p_i^2|$ . (Euclidean, or  $L_2$ , distance may be used alternatively to prevent large variance along different dimensions.)*

A typical clustering approach like  $k$ -means may fail to yield desired tradeoff because it would be difficult to specify the number of clusters, and also the radii of clusters may be affected by outliers. In order to guarantee the quality of clusters, we transform our clustering problem to the following optimization problem.

**Definition 3.** Given a collection of  $|\mathbf{U}|$  points in an  $n$ -dimensional Euclidean space (promotion cells),  $\Phi = \{P_1, P_2, \dots, P_{|\mathbf{U}|}\}$ , and a user-specified radius  $r$  ( $\geq 0$ ). A  $P_i$ -cluster is defined as  $\{P_j | 1 \leq j \leq |\mathbf{U}|, \text{dist}(P_i, P_j) \leq r\}$ , i.e., all points with  $L_1$ -distance to  $P_i$  no more than  $r$ . The problem asks for the minimum number of clusters to cover all points in  $\Phi$ .

Based on the definition, we can see that when  $r = 0$ , each cluster contains identical points while the pruning power would remain exactly the same as the promotion cube; when  $r = \infty$ , all cells are packed into a single cluster, but it is unlikely to help prune any subspace. We hence recommend to set  $r$  to about a small percentage, such as 10%, of the average gap between the aggregates of promotion cells. Unfortunately, to solve the cluster selection problem optimally turns out to be difficult given the following hardness result.

**Lemma 1.** *The cluster selection problem is NP-hard (see Appendix for proof sketch).*

Thus, we resort to a greedy algorithm as follows. First, given the set of points, compute for each point  $P_i$  the  $P_i$ -cluster. Second, iteratively select clusters until all points are covered. At each iteration, greedily select the cluster containing the maximum number of uncovered points. Finally, output the clusters selected and generate the corresponding *CluPromoCube*. This greedy algorithm has a worst-case time complexity of  $O(n \cdot |\mathbf{U}|^2)$  that is affordable for a large number of subspaces. After clusters have been selected, *CluPromoCube* will materialize their corresponding *CluCells* by merging the points. Thus, multiple subspaces can share one clustered cell, instead of storing a promotion cell for each.

## 5 Experiment

In this section we report our experimental results. First we will introduce our implementation methodology in Section 4.1. Then, comprehensive experiments will be conducted on both synthetic data set as well as real-world data set. Our performance evaluation shows that the clustering approach can achieve an order of magnitude speedup while using much smaller storage space than a baseline method does.

### 5.1 Implementation Methodology

We compare three methods in terms of query execution time and/or materialization cost. They are: (i) the clustering-based promotion cube strategy denoted by *CluPromoCube*; (ii) a naive strategy that fully materializes all results in every subspace passing a given *minsup*, which we denote using *PrecomputeAll*; and (iii) the *OnlineExec* method where queries are answered from scratch using the method described in Section 4.2. Among these three methods, *PrecomputeAll* can be regarded as the naive strategy for the materialization cost measure. On the other hand, *OnlineExec* can be considered a bottom line implementation for

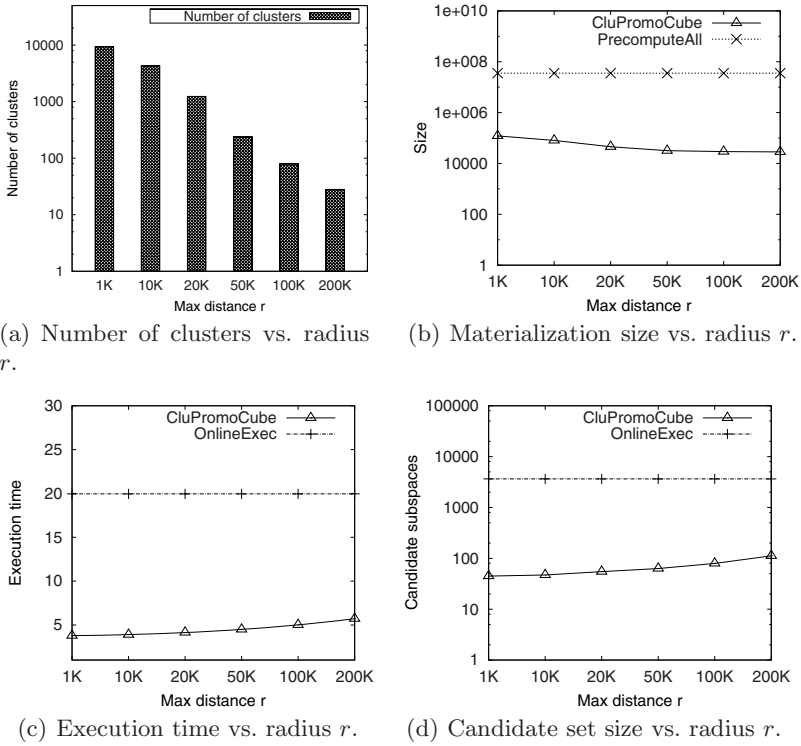


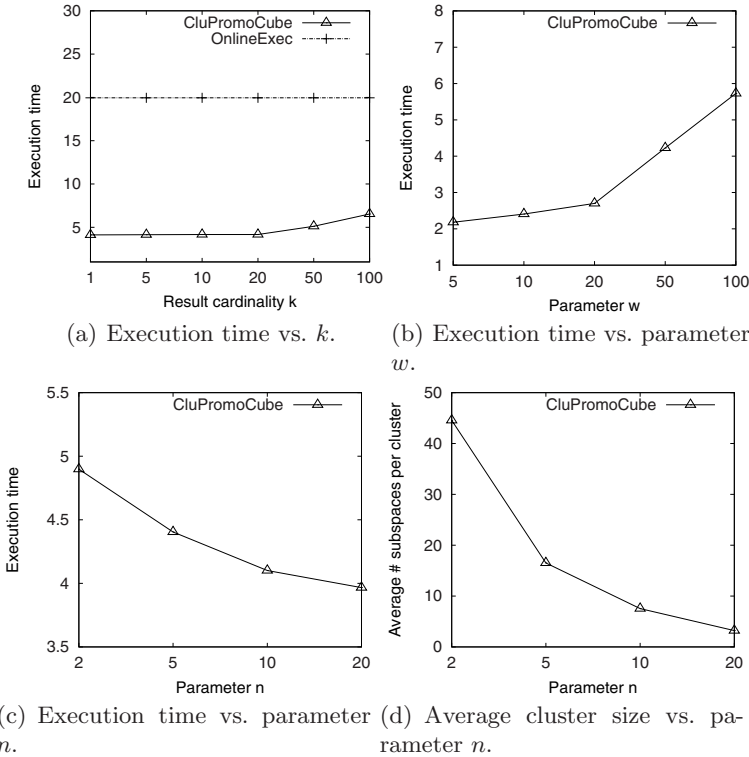
Fig. 5. Performance results on the synthetic data set

the query execution time measure as no precomputation is needed. The query execution time is measured in terms of seconds whereas the materialization cost is measured in terms of the total number of values materialized.

Our experiments were carried out on PC with a Pentium 3GHz processor, 2GB of memory, and 160G hard disk drive. The programs for the implementation were all written in Microsoft Visual C# 2008 in the Windows XP operating system. All the programs ran in the main memory and we did not count any time for loading the precomputed files before query processing since these files can be usually placed in the memory to answer multiple queries.

## 5.2 Evaluation on Synthetic Data Set

We first produced a synthetic data set using a random data generator. The data set generated has 1M rows and 8 subspaces dimensions, whose cardinalities fall in range (1, 35), and the average cardinality is 11. There are 10000 distinct objects and the score dimension contains real numbers. By default we fix  $k$  to 10 and  $\mathcal{M}$  to  $SUM$ . We also fix the minimum support threshold  $minsup$  to 2000 to filter out a large number of less interesting subspaces. For *CluPromoCube* we let  $n = 10$ ,  $w = 50$ , and  $r = 20K$  by default. All performance results are reported by averaging over 5 random target objects.



**Fig. 6.** More performance results on the synthetic data set

We first report the clustering results of *CluPromoCube*. Figure 5(a) displays the number of clusters generated when varying  $r$ , the radius parameter of the clustering algorithm. There are nearly  $10K$  subspaces passing *minsup* in total; after clustering, however, the number of clustered subspaces dramatically reduce by orders of magnitude. For example, when  $r = 50K$ , only 239 clusters are needed to cover all points, which is about  $\frac{1}{40}$  of the total number of subspaces; when  $r = 100K$ , the total number of clusters is only 79. Figure 5(b) illustrates the actual materialization cost of *PrecomputeAll* vs. *CluPromoCube* at different  $r$  values. *PrecomputeAll* need to materialize more than  $35M$  values, while *CluPromoCube* requires less than  $130K$ . Observe that, as opposed to Figure 5(a), this curve tends to be flat when  $r$  is large. For example, the size at  $r = 100K$  is only larger than the size at  $r = 200K$  by 3.3% of the latter. This is because each subspace has an overhead for maintaining a pointer to the clustered cell that accounts for a fixed amount of materialization cost. The relation between  $r$  and execution time is depicted in Figure 5(c). *CluPromoCube* is at least 3.4 times faster than *OnlineExec*. When decreasing  $r$ , *CluPromoCube* becomes faster as expected; when  $r = 1K$ , *CluPromoCube* is 5 times faster than *OnlineExec*. To justify the execution time gain, Figure 5(d) shows the unpruned number of subspaces vs.  $r$ . It is not surprising to see this curve is

correlated with the one in Figure 5(c). However, we can see that reducing space requirement only incurs a small amount of online cost, which verifies the effectiveness of the proposed method.

Now we vary the other parameters. Figure 6(a) shows the execution time vs.  $k$ . For any  $k \leq 100$ , *CluPromoCube* outperforms *OnlineExec* by at least 3 times. At  $k = 1$ , *CluPromoCube* is 4.8 times more efficient. Notice that here *CluPromoCube* has  $r = 20K$ , meaning that its total size is no more than  $50K$ . When  $k$  increases, the execution time also increases because of a weaker threshold  $R_k$ . In Figure 6(b) we show the performance by varying  $w$ . Note that varying  $w$  does not change the space requirement of *CluPromoCube*. We can see that a larger  $w$  leads to more execution time. In particular, the execution time for  $w = 5$  is more than 2 times faster than the case for  $w = 100$ . This is because when  $w$  is larger, the materialized aggregates are widely spread and thus the bound becomes less tight. Figure 6(c) further shows the performance vs.  $n$ . As  $n$  increases, the execution time becomes smaller and smaller. On the other hand, as shown in Figure 6(d), the clustering becomes “harder” in the sense that more clusters are needed when the points (promotion cells) are higher-dimensional, i.e., the average number of subspaces per cluster decreases.

### 5.3 Evaluation on the DBLP Data Set

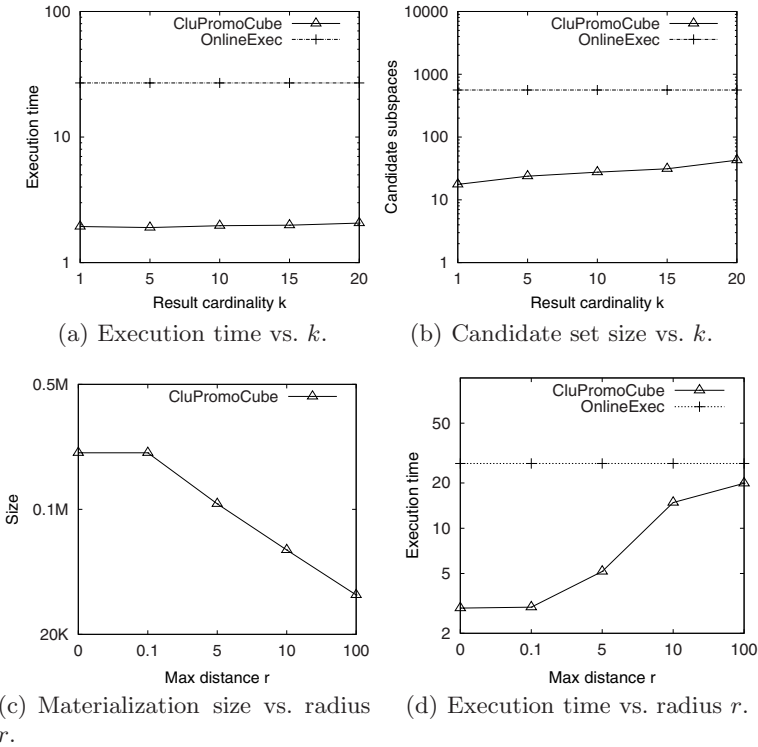
In this subsection we evaluate the *CluPromoCube* approach on the DBLP data set<sup>1</sup>. We constructed a fact table containing 1.7M tuples from it. The fact table has 6 subspace dimensions including *Venue*, *Year* and 4 other dimensions corresponding to 4 research areas. The cardinalities of the 6 dimensions vary from 50 to more than 100K. There are totally 450K distinct authors considered as objects, and we set  $\mathcal{M}$  to *COUNT*. All results are reported by averaging 5 random authors.

To evaluate the implemented methods, we constructed a *CluPromoCube* using  $minsup = 1000$ ,  $w = 5$ ,  $n = 20$ , and  $r = 0$ . Surprisingly, the overall number of 6847 subspaces can be packed into 4817 clusters even at  $r = 0$  (summarized in Table 2), meaning that each cluster represents an average of 1.42 completely same points (or promotion cells). The offline cost for this *CluPromoCube* amounts to 207,601, only 0.54% of the corresponding *PrecomputeAll*, which has a materialization cost of over 38M. Next, we compare *CluPromoCube* with *OnlineExec*

**Table 2.** Materialization cost of *CluPromoCube* with different radii  $r$ .

	<i>CluPromoCube</i>					<i>PrecomputeAll</i>
Radius	$r = 0$	$r = 0.1$	$r = 5$	$r = 10$	$r = 100$	-
Num. of subspaces	6847	6847	6847	6847	6847	-
Num. of clusters	4817	4817	1785	905	505	-
Avg. cluster size	1.42	1.42	3.84	7.57	13.56	-
Materialization cost	207,061	207,061	107,341	59,271	33,226	38,010,069

<sup>1</sup> <http://www.informatik.uni-trier.de/~ley/db/>



**Fig. 7.** Performance results on the DBLP data

in terms of efficiency. Figure 7(a) displays the results when  $k$  is varied from 1 to 20. We can see that *OnlineExec* uses constant time since it is not aware of the query parameter  $k$ , whereas *CluPromoCube* is faster by over an order of magnitude when  $k$  is not large (e.g., 13 times faster at  $k = 1$ ). Furthermore, to justify the efficiency improvement of *CluPromoCube*, we plot the number of unpruned candidate subspaces with respect to  $k$  in Figure 7(b). We can see that *CluPromoCube* need to compute much less candidate subspaces. These results therefore verify that the *CluPromoCube* approach not only performs well on real-world as well as synthetic data set.

To see how the clustering radius would affect *CluPromoCube*'s online and offline costs, we vary the parameter  $r$  in order to construct different cubes. Table 2 summarizes the clustering results. We set  $r$  to 5 values respectively, namely 0, 0.1, 5, 10, and 100, while keeping other parameters fixed. Recall that a larger value of  $r$  would lead to less clusters because each cluster is able to pack more cells. When  $r$  is set to 100, the number of clustered generated in *CluPromoCube* is 505, indicating that on average each cluster represents 13.56 subspaces. The resulting materialization cost is also significantly reduced to only 33,226; in other words, in this case *CluPromoCube* incurs only  $\frac{0.87}{1000}$  of the materialization cost of *PrecomputeAll*, or  $\frac{1}{6.2}$  of that of the *CluPromoCube* at  $r = 0$ . These results

show that there indeed exists many subspaces sharing very similar aggregates that can be clustered effectively. Note that if each aggregate value is stored in a double type using 8 bytes, the *CluPromoCube* at  $r = 0$  would consume less than 1.6MB space. We thus conclude that even for large data sets it would be feasible to maintain *CluPromoCube* completely in memory. It is also worth mentioning that the greedy clustering algorithm is in fact scalable: the offline construction time of *CluPromoCube* for any  $r$  on the DBLP data set is within 10 minutes, so we do not further analyze its complexity empirically.

In Figure 7(c), we graphically present the linear relation between  $r$  and the materialization cost shown in Table 2. We also show in Figure 7(d) the runtime of *CluPromoCube* and *OnlineExec* when varying  $r$ . As expected, the result shows that decreasing the size of *CluPromoCube* by generating less clusters would lead to more execution time, due to less effective bounding as explained earlier. Overall, although there is no universally accepted way to determine the  $r$  parameter, *CluPromoCube* is often able to achieve satisfactory performance for a wide range of parameters.

## 6 Conclusions

In this paper we have studied the promotion analysis problem and extended the promotion cube framework through a cell clustering approach in order to achieve better balance between the offline and online processing of promotion queries. We formally defined the promotion cube augmented with the clustered cell structure, as well as its complementary online algorithm. We transformed the cell clustering problem to an optimization problem and formally discussed its hardness result. A greedy algorithm is developed for generating clustered cells efficiently. By evaluating our approach against baseline strategies, we verified that the performance of our cell clustering approach is significantly better on both synthetic and real data sets. Extensions of this approach to handle other application domains such as social network analysis will be interesting directions for future study.

## References

1. Arkin, E.M., Barequet, G., Mitchell, J.S.B.: Algorithms for two-box covering. In: Symposium on Computational Geometry, pp. 459–467 (2006)
2. Chang, K.C.-C., Hwang, S.-w.: Minimal probing: supporting expensive predicates for top-k queries. In: SIGMOD Conference, pp. 346–357 (2002)
3. Charikar, M., Panigrahy, R.: Clustering to minimize the sum of cluster diameters. In: STOC, pp. 1–10 (2001)
4. Doddi, S.R., Marathe, M.V., Ravi, S.S., Taylor, D.S., Widmayer, P.: Approximation algorithms for clustering to minimize the sum of diameters. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 237–250. Springer, Heidelberg (2000)
5. DuMouchel, W., Volinsky, C., Johnson, T., Cortes, C., Pregibon, D.: Squashing flat files flatter. In: KDD, pp. 6–15 (1999)

6. Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. *J. Comput. Syst. Sci.* 66(4), 614–656 (2003)
7. Han, J., Kamber, M.: *Data mining: concepts and techniques*, 2nd edn. Morgan Kaufmann, San Francisco (2006)
8. Hochbaum, D.S. (ed.): *Approximation algorithms for NP-hard problems*. PWS Publishing Co., Boston (1997)
9. Hochbaum, D.S., Maass, W.: Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM* 32(1), 130–136 (1985)
10. Hristidis, V., Gravano, L., Papakonstantinou, Y.: Efficient ir-style keyword search over relational databases. In: *VLDB*, pp. 850–861 (2003)
11. Li, C., Ooi, B.C., Tung, A.K.H., Wang, S.: Dada: a data cube for dominant relationship analysis. In: *SIGMOD*, pp. 659–670 (2006)
12. Marian, A., Bruno, N., Gravano, L.: Evaluating top- queries over web-accessible databases. *ACM Trans. Database Syst.* 29(2), 319–362 (2004)
13. Sun, Y., Han, J., Zhao, P., Yin, Z., Cheng, H., Wu, T.: Rankclus: integrating clustering with ranking for heterogeneous information network analysis. In: *EDBT*, pp. 565–576 (2009)
14. 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)
15. Wu, T., Xin, D., Han, J.: Arcube: supporting ranking aggregate queries in partially materialized data cubes. In: *SIGMOD Conference*, pp. 79–92 (2008)
16. Wu, T., Xin, D., Mei, Q., Han, J.: Promotion analysis in multi-dimensional space. In: *PVLDB* (2009)
17. Xin, D., Han, J., Cheng, H., Li, X.: Answering top-k queries with multi-dimensional selections: The ranking cube approach. In: *VLDB*, pp. 463–475 (2006)
18. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: A new data clustering algorithm and its applications. *Data Min. Knowl. Discov.* 1(2), 141–182 (1997)

## Appendix

*Proof Sketch for Lemma 1 in Section 4.3.* We can reduce the NP-hard MINIMUM GEOMETRIC DISK COVER problem [9,8] to our problem. In an instance of the former problem, we are given a set of points on a 2d Euclidean plane and a radius  $r$ , and the goal is to compute a subset of points with minimum cardinality such that every point in the full set can be covered by a disk which centers at some point in the subset and has radius  $r$ . We transform each 2d point  $(x, y)$  to a promotion cell as  $[x + \alpha, y]$  in polynomial time, where  $\alpha$  is a large constant s.t.  $x + \alpha > y$  always holds. The radius  $r$  remains unchanged. We can prove that the optimal solutions are equal for both problem instances.  $\square$