Visual Cube and On-Line Analytical Processing of Images

Xin Jin
University of Illinois at Urbana-Champaign
xinjin3@illinois.edu

Jiawei Han
University of Illinois at Urbana-Champaign
hanj@cs.uiuc.edu

Liangliang Cao
University of Illinois at Urbana-Champaign
cao4@illinois.edu

Jiebo Luo
Kodak Research Laboratories
Eastman Kodak Company
jiebo.luo@kodak.com

Bolin Ding
University of Illinois at Urbana-Champaign
bding3@illinois.edu

Cindy Xide Lin
University of Illinois at Urbana-Champaign
xidelin2@illinois.edu

ABSTRACT

On-Line Analytical Processing (OLAP) has shown great success in many industry applications, including sales, marketing, management, financial data analysis, etc. In this paper, we propose Visual Cube and multi-dimensional OLAP of image collections, such as web images indexed in search engines (e.g., Google and Bing), product images (e.g. Amazon) and photos shared on social networks (e.g., Facebook and Flickr). It provides online responses to user requests with summarized statistics of image information and handles rich semantics related to image visual features. A clustering structure measure is proposed to help users freely navigate and explore images. Efficient algorithms are developed to construct Visual Cube. In addition, we introduce the new issue of *Cell Overlapping* in data cube and present efficient solutions for Visual Cube computation and OLAP operations. Extensive experiments are conducted and the results show good performance of our algorithms.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Data mining; Image databases; H.3.5 [**Information Storage and Retrieval**]: On-line Information Services

General Terms

Algorithms, Experimentation

Keywords

OLAP, data cube, image management

1. INTRODUCTION

With the construction of numerous large data warehouses, the industry witnesses a surge of demands of On-Line Analytical Processing (OLAP) in multi-dimensional way [6]. By offering users the ability to access data collections of any dimension subsets, an OLAP system provides the ease and flexibility for navigating data and summarizing statistics at different granularity levels and from different angles. OLAP systems have shown great success in many applications, including sales, marketing, management, and financial data analysis.

The importance of OLAP for image analysis has been recognized in applications such as remote sensing image analysis [13] and raster image analysis [1]. We believe image OLAP can also be used in image search engines, social networks and product e-commercial websites to support efficient multi-dimensional online analysis of images.

Data cube [5] is the workhorse for OLAP. Because of the distinguished nature of visual information, the data cube model for images should be different from the existing data cube models, such as [11] [7] [10] [2]. To design a good image data cube model and support efficient OLAP of images, there are four fundamental issues that need to be addressed: (1) design useful cube dimensions to support multi-dimensional organization of the images; (2) design useful measures to support user analysis; (3) efficiently construct data cube; and (4) design efficient OLAP operations.

We provide comprehensive and efficient solutions for image data cube regarding all the above four fundamental issues. We propose a general model of *Visual Cube* for OLAP of images. Visual Cube not only summarizes statistics information, but also helps users navigate and analyze the images efficiently.

Related studies [19] [13] partially deal with the image dimension issue in a traditional data cube way. However, there are open problems if considering the special properties of images. For example, how to design dimensions on image tags or major colors? The challenge is that an image can have more than one tag or major color, this is different from traditional data cube where a record only has one unique value for a dimension. We introduce two types of schemes, MDS (Multi-Dimension Scheme) and SDS (Single-Dimension Scheme), and will show that SDS is the practical scheme to build dimensions on such information for large image datasets. However, SDS scheme introduces the *Cell Overlapping* phenomenon. We will analyze the impact of overlapping and propose efficient solutions in Visual Cube computation. We also introduce new OLAP operations considering the overlapping and develop efficient algorithm to handle it.

The dimensions, such as Time, Location and Tags, in Visual

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CIKM'10, October 26–30, 2010, Toronto, Ontario, Canada.
Copyright 2010 ACM 978-1-4503-0099-5/10/10 ...$10.00.

Cube provide a good way of image organization. However, they are not enough for navigation and analysis of large image data, because many images could share the same dimension value. For a query like Q: ("Year = 2006"), there could be over thousands of images taken in that year. Simply return all those large number of images without a good organization or summarization would be too overwhelming.

To solve this problem, we introduce clustering structure as a measure for Visual Cube. The idea is to perform content-based image clustering (based on image features such as color/texture/shape) to group visually similar images together, and then choose central images from the clusters as the representatives shown to the user. When interested in any representative image, the user can click on it and explore into related similar images. As addressed in [3] [17], clustering technique is a natural fit for exploring and describing large image repositories. We also propose other types of measures for Visual Cube and provide a general categorization.

Based on the Visual Cube model, we study how to compute the cube efficiently. For the clustering structure measure, a straightforward algorithm is to aggregate image ids at the cells and perform clustering for a cell based on the content features of all the images in the cell. In this approach, the clustering of a cell is performed independently of other cells.

We also propose more efficient methods. The basic idea is to utilize the already computed clustering structures of the lower level cells to help the clustering of images in a higher level cell. For example: Given two lower level cells c_1 (Tag = "tree", Year = "2006") and c_2 (Tag = "tree", Year = "2007"), and the higher level cell c_3 (Tag = "tree", Year = *), where * means cell c_3 is aggregated on the Year dimension. (Refer to Fig. 2 for a real similar example in our system.) Suppose clustering are performed in cells c_1 and c_2, we want to use the clustering structures to help efficiently perform clustering in the higher level cell c_3.

The motivation is that since the images in the higher level cell is a combination of the images in the lower level cells, if images are similar and grouped together by clustering in a lower level cell, they will have high possibility to be also clustered together in the higher level cell. We assume the images in different cells are clustered based on the same type of image feature. So we can treat any set of clustered images as a meta-point, and directly aggregate them to a higher level without doing clustering on the individual images again. In this way, the clustering on higher level cells will be performed based on meta points, thus will be much faster than performed on the individual images.

The above method is a naive approach to achieve better efficiency, but may degrade the clustering quality, we will provide deeper analysis and present clustering aggregation algorithms to achieve good balance on speed and clustering quality. Consideration and solution on the *Overlapping* case will be also presented. In addition, dynamic aggregation selection will be proposed in this paper to select the best lower level cells for aggregation and improve the performance.

To support multiple features, such as color, texture and shape, in Visual Cube, we can save different clustering structures at a cell, each for color, texture and shape, respectively. When aggregating from lower level cells to a higher level cell, we perform three clustering aggregations, each based on the clustering structures of the same image feature type. In this way, the system provides user with the flexibility of choosing his/her interested features to navigate and analyze the images.

Based on Visual Cube, we can support efficient OLAP on images, as shown in Figure 1. The dimensions include Time (Year, Month, Day), GPS locations (two levels of latitude and longitude), Tag and Color. The measures are Count, and the representative images with one from each cluster. By clicking on any image, we can further explore into related similar images.

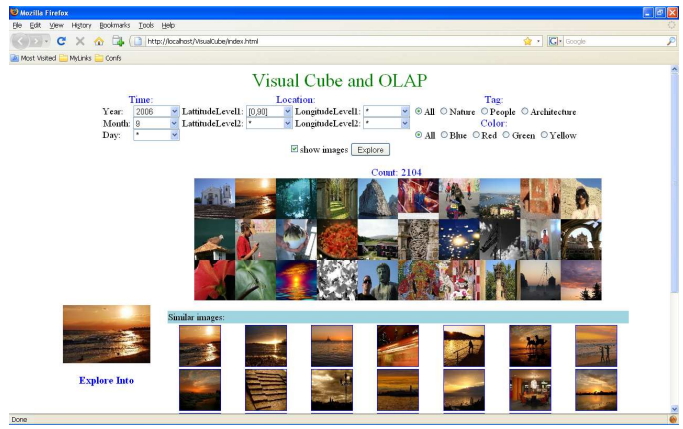


Figure 1: Visual Cube and OLAP.

Figure 2 shows two example OLAP operations, Roll-up and Drill-down, on the Lattitude (Lat) dimension. The images in the cells are arranged in a clustered manner by clustering (here we show all images instead of representative images).

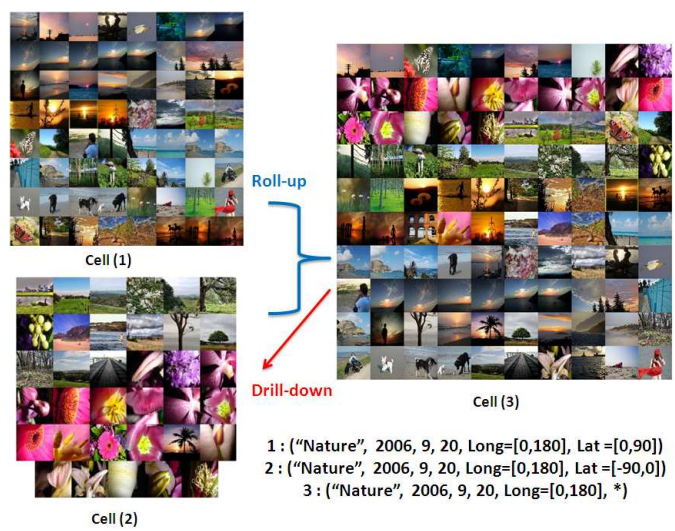


Figure 2: OLAP operations on the Lattitude (Lat) Dimension.

Figure 3 shows the system framework for Visual Cube and OLAP for images.

This paper is organized as follows: Section 2 outlines the design of Visual Cube. Section 3 presents Visual Cube construction algorithms. Section 4 discusses Visual OLAP operations. Section 5 presents experimental results. Section 6 concludes this paper.

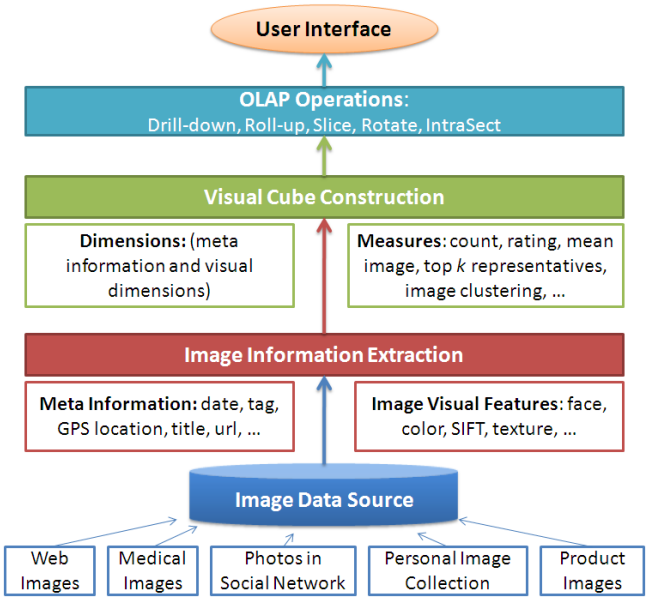


Figure 3: Visual Cube and OLAP system framework.

2. VISUAL CUBE STRUCTURE

Given a set of images, we extract their meta information (such as title, date, GPS location, tags and url) and visual features (such as color/texture/shape), from which the raw image database will be built. Table 1 shows an example image database with extracted color histogram $H_i \in \mathbb{R}^d$.

Table 1: Example raw image database

Id	Date	GPS	Tags	ColorHist
1	9/10/2007	(31.1, 78.5)	sunset	H_1
2	2/5/2007	(45.2, 28.9)	girl	H_2
3	4/8/2007	(35.7, 128.9)	girl, sunset	H_3
4	6/15/2008	(−78.5, 135.4)	sunset	H_4
5	10/1/2008	(−68.5, 35.1)	sunset	H_5

Based on the raw image database, given a multidimensional data model with indicated dimensions A_i, we can build Visual Cube. A cell c is represented as $(a_1, a_2, \ldots, a_n, M)$, where n is the number of dimensions and M is the measure. When $a_i \in \{*\}$, the cell is aggregated on dimension i.

Without losing generality, take a 4-D cube as an example, with dimensions A, B, C, D. Suppose we aggregate cuboid AB from ABC, then we call AB as the *target cuboid*, and ABC the *supporting cuboid*. A cell in the target cuboid is a *target cell*. A cell in the supporting cuboid is a *supporting cell*. A target cuboid could have more than one *candidate supporting cuboid*. For example, AB can be aggregated from either ABC or ABD.

2.1 Dimensions

Two types of dimensions are built for Visual Cube: (1) Meta Information Dimensions, such as date, title, file name, owner, URL, tag, description, interestingness, license and GPS location; and (2) Visual Dimensions (based on image visual features), such as image size, major colors, face dimension (indicating the existence of faces), color/texture histogram.

The tag or major colors dimension is especially challenging for Visual Cube, because an image may have multiple tags or major colors. We introduce two schemes to build such type of dimension: Multiple Dimension Scheme (MDS) and Single Dimension Scheme (SDS).

Table 2 shows cuboids built from the example image database in Table 1 using the MDS scheme (a) and the SDS scheme (b), respectively. In the MDS scheme, the cuboid has three dimensions: Year, Sunset and Girl; while in the SDS scheme, the cuboid has only two dimensions: Year and Tag. The two cuboids share the same information, but the latter is more compact. Table 2 (c) shows the 1-D cuboid aggregated to the Year dimension.

Table 2:

(a) MDS scheme, a 3-D cuboid

Year	Sunset	Girl	RidS	Ct
2007	1	0	{1}	1
2007	0	1	{2}	1
2007	1	1	{3}	1
2008	1	0	{4, 5}	2

(b) SDS scheme, a 2-D cuboid

Year	Tag	RidS	Ct
2007	sunset	{1, 3}	2
2007	girl	{2, 3}	2
2008	sunset	{4, 5}	2

(c) A 1-D cuboid

Year	RidS	Ct
2007	{1, 2, 3}	3
2008	{4, 5}	2

The MDS scheme treats each tag as one dimension, with value 0 or 1, indicating whether the tag is annotated to the images in the cell. There is only one level available for each dimension. In this scheme, no overlapping exists between cells, i.e., no image belongs to more than one cell.

The disadvantage of the MDS scheme is that since the overall number of unique tags in the image collection is usually large (over tens of thousands), there would be too many dimensions. We know the number of cuboids $|B|$ is at least exponential to the number of dimensions $|D|$. Since $|D| = |P| + |T|$, where $|P|$ is the number of non-tag dimensions, and $|T|$ is the number of tags, we have $|B| = 2^{|P|+|T|}$. It is unrealistic to build Visual Cube using this scheme when there are many tags.

To solve the problem of MDS, the SDS scheme builds a single dimension for all the tags. The tag hierarchy could be manually or automatically built in this scheme. The advantage of SDS is that there is only one tag dimension, so $|B| = 2^{|P|+1}$, which is independent with $|T|$ and is much smaller than the MDS scheme.

Overlapping OLAP. We have shown that the SDS scheme is significantly more compact than MDS and is the practical choice for building Visual Cube with many tags. However, it introduces a problem which we call *Cell Overlapping*. If an image has multiple tags, i.e., multiple values in the tag dimension, it will appear in multiple cells. For example, in Table 2 (b), image 3 belongs to both the first and second cells.

We call the dimension which causes cell overlapping as *Overlapping Dimension*. Examples include tag, color (an image may have several major colors), topic (in text/topic cube, a document may belong to several topics), etc. We call an OLAP system containing overlapping dimensions *Overlapping OLAP*.

In Overlapping OLAP, even the simplest measure *Count* can not be directly aggregated from any lower level cuboid on the overlapping dimension. For example, if we directly

add the *Count* of the first and second cell in cuboid from Table 2 (b), the *Count* for cell "Year = 2007" of the aggregated 1-D cuboid in Table 2 (c) will be 4 which is not correct.

2.2 Measures

We introduce four categories of measures for Visual Cube. The first category is similar to the measures used in traditional data cube, however, the other three categories are unique for Visual Cube. (1) Summarized information: count (number of images in a cell), max/mean/min/std rating, etc; (2) Summarized image feature: mean image, average color histogram, major colors, etc; (3) Subset of the images: top-k representative images (by clustering images based the image features (such as color histogram) and choose the central images as representatives), etc; and (4) All the images in the cell, but in some kind of organization, such as a ranked list (like in image search engines) and clustering structure, which is computed from the image features and provides user better overview of the images.

We define the **clustering structure** G_c of cell c to consist of the cluster centers and a set of member image ids for each cluster. An image can be represented as a feature point $p \in \mathbb{R}^d$. By employing the clustering structure as a measure, Visual Cube enables users to not only navigate the images freely, but also find interesting phenomena from image groups.

We define the **clustering quality** of cell c as the average distance between points to its cluster center, i.e., $Q(c) = \sum_{i=1}^{|c|} D(p_i, center(p_i))/|c|$, where $|c|$ is the number of images in the cell. The overall clustering quality of cube B is $Q(B) = \sum_{i=1}^{|B|} Q(c_i) * w(c_i) / \sum_{i=1}^{|B|} w(c_i)$, the weighted average quality of all its cells. To avoid large size cells dominating the cube quality, we choose $w(c) = |c|^{0.5}$ to lower the impact of size increasing. Given a certain number of clusters, smaller value of Q means the images in the same cluster are more similar to each other, which indicates better clustering quality.

Semi-Holistic Measure: In traditional data cube, there are three standard types of measures: *distributive*, *algebraic* and *holistic*. A *holistic* measure is hard to aggregate because it needs to access all the corresponding records in the raw database to compute the value. We introduce a new type of measure for Visual Cube: *semi-holistic*, which only needs to access part of the corresponding records in the raw database to compute the value. We will show in this section how *semi-holistic* works and why it is more efficient than *holistic*.

Aggregate Measures in Overlapping OLAP. We use **RidSet** (the set of record ids of the cell. RidS for short) to help the aggregation of non-distributive measures. RidSet is *distributive*. To aggregate it, the basic operation involved is *Union* of the sets. We discuss two examples to demonstrate its usage in computing other types of measures.

(1) **Count**. In traditional OLAP, this measure is *distributive* and can be aggregated directly by adding up the counts from supporting cells. However, in overlapping OLAP, this measure becomes *Algebraic*. We need to first aggregate RidSet, and then use its size as count.

(2) **Mean**: There are two ways to aggregate this kind of measure: *holistic* and *semi-holistic*. The *holistic* method works as follows: (1) compute the union of the RidSet's of supporting cells sequentially $U = \bigcup_{i=1}^{S} RidSet_i$, S is the number of supporting cells; (2) access values in U; and (3) compute mean as $Mean_{agg} = \sum_{i=1}^{|U|} value_i * Count_{agg}^{-1}$.

The *semi-holistic* method works as follows: (1) aggregate count; (2) compute the intersection of the RidSet's of the supporting cells sequentially $I = \bigcap_{i=1}^{S} RidSet_i$, and count the times an id repeats, so $(repeat_i - 1)$ is duplicate number; (3) access original values of records in I; and (4) compute aggregated mean using the following formula:

$$Mean_{agg} = \frac{\sum_{i=1}^{S} Mean_i * Count_i - \sum_{i=1}^{|I|} value_i * (repeat_i - 1)}{Count_{agg}} \quad (1)$$

Table 3 summarizes each measure's aggregation property in traditional (non-overlapping) data cube and overlapping data cube.

Time and Space Efficiency. Instead of literally saving the set of ids for RidSet, we can save it in a compressed form by techniques such as δ-code. For time efficiency consideration, we can save the RidSet as *bitmap*, thus the set computation can be efficiently performed by bitmap operations. To save space, bitmap can be compressed by techniques such as Word-Aligned Hybrid (WAH) [18].

Table 3: Aggregation type of measures in traditional OLAP (T-OLAP) and Overlapping OLAP (O-OLAP).

Measures	T-OLAP	O-OLAP
RidSet, Max, Min	distributive	distributive
Count	distributive	algebraic
Sum	distributive	semi-holistic
Mean	algebraic	semi-holistic
Median, Mode	holistic	holistic
Clustering Structure	not discussed	(semi-)holistic

3. VISUAL CUBE CONSTRUCTION

We focus on full materialization for Visual Cube construction since it enables the fastest OLAP operations and sets the algorithmic foundation for partial materialization. Partial materialization helps reducing storage especially when the cube is in high dimension, and we leave this for future work.

3.1 General Framework

We propose a general framework for Visual Cube construction as follows: (1) based on the raw image database, build the base cuboid. For multi-value attributes, such as *tags* and *major colors*, adopt the SDS scheme and expand all combinations of the values to make sure each cell is unique; (2) calculate measures for each base cell; and (3) perform aggregation selection and aggregate measures for cells in higher level cuboids. We choose bottom-up approach (i.e., from base cells up) and leave the top-down approach for future work.

Algorithm 1 shows the framework for the Visual Cube construction with *clustering structure* measure (other measures can also be computed in this framework). The popular k-Means [16] algorithm is used for the basic clustering. Many other fast clustering algorithms, such as GAD [8], are also applicable. If the user is interested in soft partitioning, we can use EM algorithm and aggregate high possibility assignments as hints to help clustering on higher level cells.

3.2 Dynamic Aggregation Selection

We propose the idea of dynamic aggregation selection in this section. Aggregation selection answers this question of

Algorithm 1 Visual Cube Construction Framework

Input: IDB, the image database
Output: A Visual Cube with clustering measure

```
build base cuboid from IDB
for each cell c in the base cuboid do
    if Count_c > k do clustering(c)
end for
for each cuboid b at higher level do
    for each cell c in b do
        b'=AggregationSelection(c)
        S := the supporting cells from b' for c
        G_c = AggregationClustering(S)
    end for
end for
```

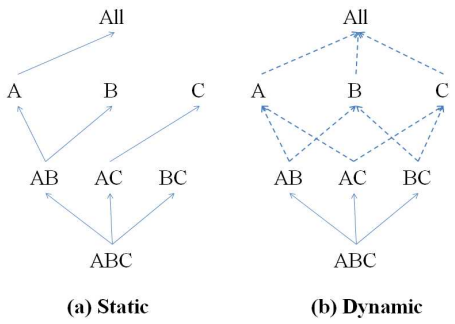


Figure 4: Aggregation selection schemes.

choosing the best candidate supporting cuboid for aggregation. For example, if the cells in cuboid AB can be aggregated from both ABC and ABD, which one to choose to get better performance?

Traditional data cube computation algorithms, such as MultiWay [20], adopt static aggregation selection scheme which follows a global order based on decreasing dimension cardinality. More specifically, the candidate supporting cuboid whose aggregation dimension has minimum cardinality is selected for a target cuboid. This is widely used in traditional data cube computation. However, in Visual Cube, especially for the computation of the clustering measure, the aggregation selection problem becomes complex. Because different choices may result in different computation time and quality.

In the static method, cells in the same cuboid always select the same cuboid to aggregate from. However, this is not the optimal scheme, because different cells in the same cuboid may have the best set of supporting cells from different candidate cuboids.

Based on the above observation, we go deep into the cell level, and dynamically select the best supporting cuboid for each individual cell. We calculate a selection score $SS(b, c)$ of a supporting cuboid b w.r.t the target cell c, and choose the one with the best score. There are several factors: C (the cardinality of the aggregation dimension), S (number of supporting cells in b for c), Q (the overall clustering structure quality of the supporting cells) and O (overlapping property of the supporting cells). We use two schemes DynCN ($SS(b, c) = -S$) and DynCQ ($SS(b, c) = -Q$) to demonstrate the advantage of dynamic aggregation selection for Visual Cube.

For the dynamic schemes, the cells in the same targeting cuboid may choose different supporting cuboids for aggregation. The advantage is that it's able to select the best supporting cuboid for each individual target cell. Fig. 4 shows the difference between static and dynamic aggregation selection.

3.3 Clustering Structure Aggregation

In order to compute clustering structure for the cells, a naive aggregation approach is to perform clustering independently at each cell, which we call Independent Aggregation (**IA**). It works as follows: For each target cell, aggregate RidSet from the supporting cells, access the corresponding image features based on the RidSet and perform clustering based on the image features.

In the IA algorithm, the clustering structures of the supporting cells are ignored when computing the target cell. The disadvantage is that it is time consuming to access and compute the high dimensional image feature of all the images for all the cells. For OLAP query, if the target cell is not materialized, this approach cannot provide real-time response.

In the following sections, we present more efficient cube aggregation algorithms with different trade-offs between speed and clustering quality. The general idea is to utilize the existing clustering structures of the supporting cells to improve the efficiency for data cube computation at a target cell. This is related to distributive computing [12].

3.3.1 Meta-Point Aggregation (MP)

A *meta-point* (or meta-cluster) is defined as a center or centroid point that represents a set of similar individual points. We introduce the Meta-Point aggregation (**MP**) algorithm. The basic idea is to directly use clusters from supporting cells as meta-points to be clustered in the target cell.

MP works as follows: Beginning with the base cuboid, perform clustering for each base cell. Each cluster in the base cells forms a meta-point. To calculate the clustering structure of any higher level cell, we first aggregate the meta-points from the supporting cells of the selected supporting cuboid using *union* operation, and then perform clustering on the meta-points. Points in the same meta-cluster are assigned to the same center at the higher level cell. Overlapping removal is finally performed. Algorithm 2 describes the aggregation part of the MP algorithm. Figure 5 shows an example.

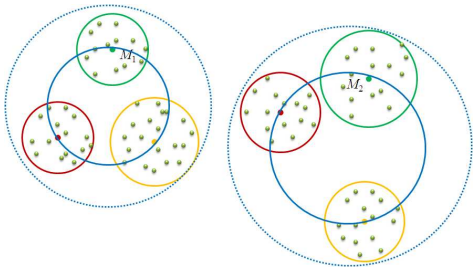


Figure 5: Meta-Point Aggregation. Aggregated from three supporting cells (red, green and yellow, respectively). $k = 2$ **for the target and supporting cells. Blue circles are the aggregated clusters for the target cell.**

Overlapping Removal. Fig. 6 shows an overlapping situation. Point P_1 appears in both meta-point M_1 and M_2, which are assigned to C_1 and C_2, respectively. In order to remove duplicates, we need to assign the point only to the best match cluster.

Algorithm 2 Meta-Point Aggregation (MP)

Input: S, the set of supporting cells
Output: tc, the target cell

```
M = ∅    // initialize meta points M
for each supporting cell s in S do
    if s has clustering structure then
        for each cluster g of s do M ⋃= g.center
    else
        for each member point p of s do M ⋃= p
end for
G_M = clustering(M)
for each cluster g_i of G_M do
    G_tc(i).members = ∅
    for each member point m of cluster g_i do
        if m is a meta-point then
            G_tc(i).members ⋃= m.members
        else G_tc(i).members ⋃= m
    end for
end for
tc = removeOverlapping(tc)
```

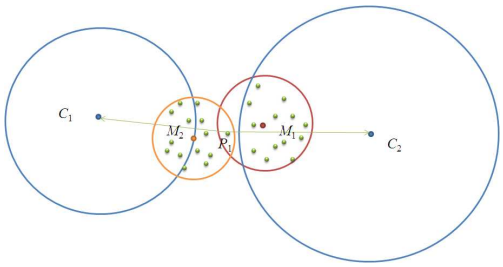


Figure 6: Situation where overlapping exists

To remove overlapping efficiently, instead of checking each pair of clusters to find overlappings, we perform fast linear complexity checking by sequentially scanning the clusters only once and accumulating the cluster ids for each point of the cell. Then, if a point belongs to more than one cluster, decide the best match cluster and assign to it; otherwise, stick to its original cluster. Algorithm 3 describes the overlapping removal procedure.

Algorithm 3 Remove Overlapping

Input: c, the target cell
Output: c', the cell with overlapping removed

```
initialize the candidate clusters of each point p in c as
    p.clusters = ∅
for each cluster g in c do
    for each point p in g do p.clusters ⋃= g
end for
for each point p in c do
    if |p.clusters| > 1 then
        i:= the best match cluster among p.clusters
        assign p to cluster i
    else
        keep p's original cluster assignment
end for
```

There are two methods to decide the best match cluster for an overlapping point. One method accesses the point's original feature and calculates its distances to all the related clusters to find the nearest one. The other method directly uses the distance of the meta-point to its cluster center, and chooses the minimum distance center. The latter approach is more efficient because it does not access the original feature and no new distance calculation is involved, but it may degrade the clustering quality because it cannot guarantee always finding the correct the nearest center.

Time efficiency of MP. The MP algorithm is the most time efficient, because it solely depends on the already computed clustering structure from the supporting cells and avoids accessing the original image features.

Clustering quality of MP. MP performs clustering on meta-points instead of individual points. It assumes that all the member individual points of a meta-point are closest to the center of the cluster which the meta-point belongs to. However, the assumption does not hold in some real applications and results in low clustering quality. Fig. 7 shows an example. C_1 and C_2 are the centers of two clusters, 1 and 2, respectively. M_1 is the center of a meta-point, P_1 is a member point of the meta-cluster. Let $D()$ denotes the distance function, we can see that $D(M_1, C_1) > D(M_1, C_2)$, so this micro-point is assigned to cluster 2. However, when we decluster the meta-point and reveal point P_1, we get $D(P_1, C_1) < D(P_1, C_2)$, the point should be assigned to cluster 1 instead of cluster 2.

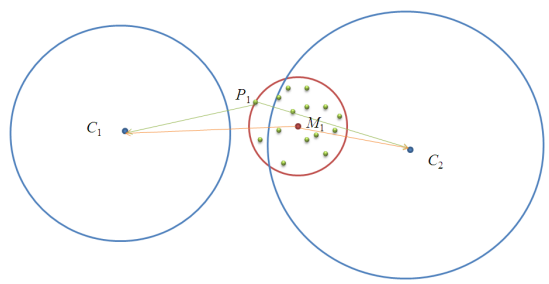


Figure 7: Situation when cluster assignment conflicts for meta-point and its member individual point.

3.3.2 *Partial-Declustering Aggregation (PD)*

To improve the quality of MP, we introduce Partial Declustering aggregation (**PD**). The idea is to find boundary meta-points to decluster and release the individual points associated with them. For each released individual point, calculate its distance to each cluster center, find the closed center and assign to it. In this way, we can find the true nearest center of such boundary points and thus improve the clustering quality.

A **boundary meta-point** locates at boundary areas and has high chance to contain many individual points which should be re-assigned to another cluster. We define a criteria to find boundary meta-points for declustering.

Take 2-D feature space as an example, as shown in Fig. 8. Assume that a meta point M_1 lies between its nearest cluster center C_1 and its second nearest cluster center C_2. Build a Cartesian coordinate system in the following way: set C_1 as the origin, set the line which goes through the origin to C_2 as the X-axis. Given the radius r of the meta-point, distance $d_1 = D(M_1, C_1)$ and distance $d_2 = D(M_1, C_2)$, we calculate m as the middle of C_1 and C_2, E as to what extent the meta-point

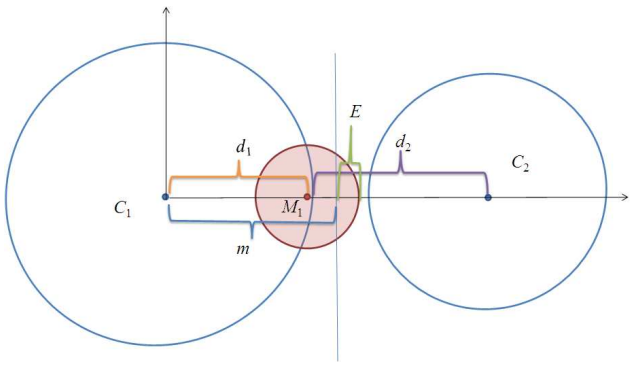


Figure 8: Demonstrate the declustering criteria.

goes beyond the middle:

$$m := \frac{d_1 + d_2}{2},\ E := d_1 + r - m \tag{2}$$

Criteria for de-clustering:

$$\frac{E}{2r} > \lambda \tag{3}$$

Parameter λ is an approximate estimation for how many percent of member individual points go beyond the middle line, in which case the de-clustering is needed.

Algorithm 4 presents the partial-declustering aggregation procedure.

Algorithm 4 Partial-Declustering Aggregation (PD)

Input: S, set of supporting cells
Output: tc, target cell

```
1. Same as step 1 to 13 of Algorithm 2
2. for each cluster g_i of G_M do
3.     G_tc(i).members = ∅
4.     for each member point m of cluster g_i do
5.         if m is a meta-point then
6.             calculate r, d1, d2
7.             if m satisfies the declustering criteria then
8.                 for each individual point p of m do
9.                     j = NearestCenter(p, G_M)
10.                    G_tc(j).members ⋃ = p
11.            else G_tc(i).members ⋃ = m.members
12.        else G_tc(i).members ⋃ = m
13.    end for
14. end for
15. tc = removeOverlapping(tc)
```

PD gets higher clustering quality than MD by declustering some *hard* clusters, with the expense of spending more computation on those released individual points.

3.3.3 Full-Declustering Aggregation (FD)

Instead of selecting some meta-clusters to decluster, we can perform Full-Declustering aggregation (**FD**) which declusters all meta-points to guarantee every individual point will be correctly re-assigned to its true nearest cluster, thus achieving higher clustering quality than PD, with the expense of possibly more computation time because more meta-clusters are declustered to be analyzed.

To achieve even better quality, we can allow the points to be reassigned according to updated centers and reiterate the procedure. We called this extension as FD with Re-assignment (**FDR**). Parameter r controls the number of reassignment iterations.

4. VISUAL OLAP OPERATIONS

Based on the Visual Cube, OLAP operations, such as Drill-down, Roll-up and Slice, can be efficiently supported. We also introduce a new type of operation regarding the overlapping phenomenon in Visual OLAP and give an efficient solution.

Take Slice as an example. It performs a selection on one dimension of the cube. If the selection is on multiple values, (for example, the user is interested in both 2009 and 2010 on the Year dimension), for non-overlapping OLAP, only OR operation can be performed, while AND operation is not discussed in the literature because it always generate NULL since no record exists in multiple cells. However, for overlapping OLAP where a record can exists in multiple cells, the AND operation is meaningful, an example query is: (Tag="sunset" AND Tag="girl"). OR operation is the *Union* aggregation of the selected cells, while AND operation is the *Intersection* aggregation of the selected cells. For the clustering structure measure, OR operation can be supported by any of the clustering aggregation algorithms proposed in this paper.

We call intersection aggregation of cells within the same cuboid on the same subset of dimensions as **IntraSect** operation. A basic algorithm for such operation is as follows: (1) detect related records by intersection of the RidSet's of the selected cells; (2) perform Drill-through to the raw database to access the original features of those records; and (3) do image clustering based on the content features.

The above algorithm is not efficient to support online query when the related images are in large number: firstly, it requires accessing the raw database which takes time; secondly, it performs clustering on original returned images and the features, the number of images could be large and the features are usually in high dimension. So the procedure cannot be finished in short time.

To deal with the above problem, we can design method to directly utilize the existing clustering structures of the cells and perform clustering on reduced size without considering all the original features accessed from the raw database.

5. EXPERIMENTAL EVALUATION

This section presents extensive experimental evaluation for the Visual Cube computation algorithms. Experiments were conducted on a PC with a Intel Pentium(R) D 3.4GHz CPU and 4GB RAM, running Windows XP.

5.1 Dataset

Our dataset consists of over 114,000 Flickr [4] images downloaded via using Flickr API. We extract information such as the time, tags, GPS locations, and image features to build up the raw database, and compute the Visual Cube. We extract 75-dimension LAB features [9] and use Principle Component Analysis (PCA) [14] to reduce the dimension size to 30. Note that our framework is flexible to other image features.

For time performance evaluation, we use the Speedup of algorithm A over baseline B: $Speedup(A, B) = T_B/T_A$, where T_A is the execution time of using A to compute the Visual Cube and T_B is the execution time of using B.

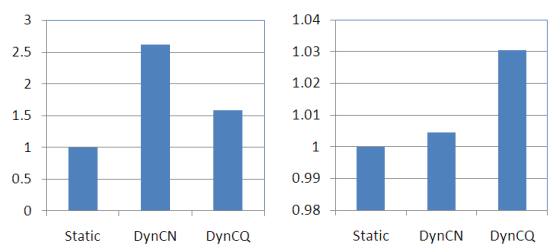
For clustering structure quality evaluation, we use Quality Ratio (QR): $QR(A, B) = Q_B/Q_A$, where Q_B is the quality of the cube computed by the baseline algorithm B. Since the value of Q is the smaller the better, if $QR > 1$, algorithm A gets better clustering quality cube than the baseline algorithm B.

5.2 Results

We compare the performances of Visual Cube construction algorithms, including the aggregation selection schemes (Static, DynCN and DynCQ) and clustering measure aggregation algorithms: the *baseline* Independent Aggregation (IA), Meta-Point aggregation (MP), Partial-Declustering aggregation (PD), Full-Declustring aggregation (FD) and Full-Declustering with Re-arrangement (FDR).

5.2.1 *Performance of Aggregation Selection Schemes*

Fig. 9 shows the performance of aggregation selection schemes: Static, DynCN and DynCQ. The Speedup and QR are calculated using Static as the baseline. We set $n = 20000, d = 9, k = 30$, and the results are generally similar for other settings. We use MP as the clustering aggregation algorithm.



(a) Time (Speedup) (b) Quality (QR)

Figure 9: Performance of Aggregation Selection Schemes. Y-axis denotes the measure.

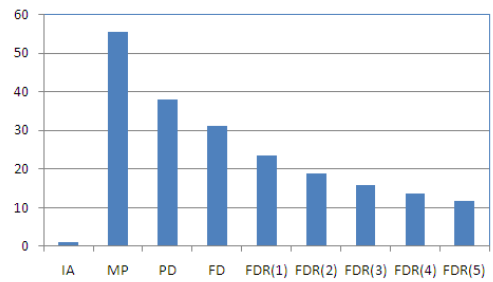
The result shows that compared to the traditional static method, the dynamic aggregation approaches DynCN and DynCQ get better performance in time and quality. DynCN achieves a 2.62 times Speedup over Static. The reason is that for any target cell, DynCN is able to always find the minimum number of supporting cells for aggregation. Usually, a smaller number of supporting cells result in lower time complexity. DynCQ achieves better quality than Static, because it is able to always choose the best quality supporting cells for computing the target cell.

We use DynCN as the default aggregation selection scheme for all other experiments to make sure they are in the same setting for comparison of other aspects.

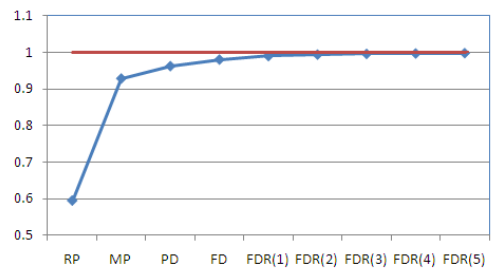
5.2.2 *Performance of Aggregation Algorithms*

Fig. 10 shows the performances of the clustering structure aggregation algorithms: MP, PD, FD and FDR. The parameter r of FDR varies from 1 to 5. The evaluation metrics are computed using IA as the baseline. To prove the usefulness of doing clustering, we also compare with Random Partition (RP).

The result shows that the clustering aggregation based algorithms get much better time performance than IA. MP achieves the highest speedup but lower quality because its computation is soley based on meta-points. The de-clustering idea makes the quality better and still get very high speedup. RP shows very poor quality compared with baseline, and



(a) Time performance (Speedup)



(b) Quality performance (QR)

Figure 10: Performances of the aggregation algorithms. Y-axis denotes the performance measure. X-axis denoted the algorithms. ($n = 10000, d = 6, k = 40$)

proves that there does exist hidden clustering structure with similar images in Visual Cube, and performing clustering (neither directly or aggregated) can find such similar images. PD gets better quality(improve to be over 96% approximation) than MP by de-clustering some boundary meta-points to make the cluster assignment for the individual points more accurate, while FD further improve the quality (to be over 98%) by de-clustering all meta-points. Finally, by performing several re-assignments, FDR achieves the highest quality.

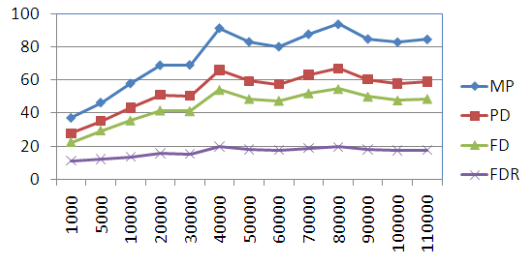
For the FDR algorithm, with the increase of r, the computation time increases, but the clustering quality also increases. When $r = 5$, it can achieve almost the same quality as the baseline. We set $r = 5$ as the default value for the FDR algorithm for other experiments.

5.2.3 *Performance w.r.t the Number of Records*

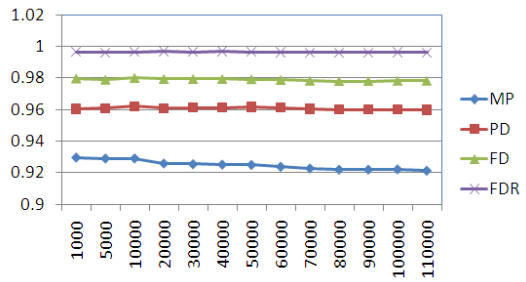
Fig. 11 shows the performances of aggregation algorithms w.r.t n (the number of records) which varies from 1000 to 110,000. The baseline algorithm is IA. The four aggregation algorithms are much faster than IA. The best speedup is 94 times faster by MP at $n = 80,000$. With the increase of n, the speedup also increases, the reason is that the algorithms are initially based on the meta-points instead of individual points, with a given number of clusters, the meta-points will tend to represent more individual points and thus reduce the computation complexity. For the same reason, the quality tends to become lower but the decrease is very slightly since in most cases a meta-point is a good representative for its individual member points.

5.2.4 *Performance w.r.t the Number of Dimensions*

Fig. 12 shows the performances of the aggregation algorithms w.r.t d (the number of dimensions) which varies from

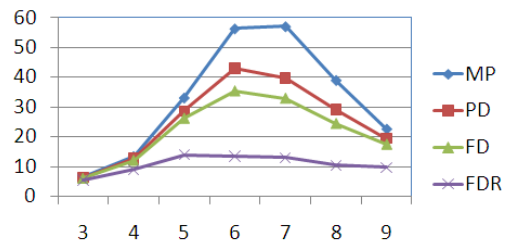


(a) Time performance (Speedup)

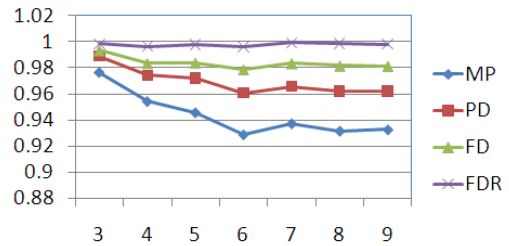


(b) Quality performance (QR)

Figure 11: Performances of aggregation algorithms w.r.t n, the number of records (X-axis). Y-axis denotes the performance metric. ($d = 6, k = 20$)



(a) Time performance (Speedup)



(b) Quality performance (QR)

Figure 12: Performances of algorithms w.r.t d, the number of dimensions (X-axis). Y-axis denotes the performance metric. ($n = 10000, k = 20$)

3 to 9. The baseline algorithm is IA. The four aggregation algorithms always beat IA in time performance. The best performance of 57 times speedup is achieved by MP when $d = 7$. When the d is smaller, the number of cells which perform aggregation based on meta-points becomes also smaller among all cells, so the speedup is not very high (but still over 5 times); however, correspondingly, the overall quality becomes very high since more computations are performed directly on the individual points.

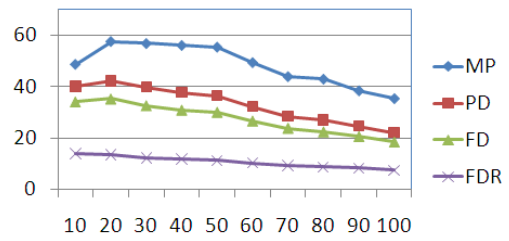
For 8 and 9 dimensions, the time performances of the algorithms drop (but still several times faster than baseline). There are two reasons: firstly, since we fix the number of records, with large dimensions, many cells are small and thus the advantage of using meta-point instead of individual point becomes relatively less significant; secondly, the last two dimensions are overlapping dimensions, some extra computations are needed for removing overlapping.

5.2.5 *Performance w.r.t the Number of Clusters*

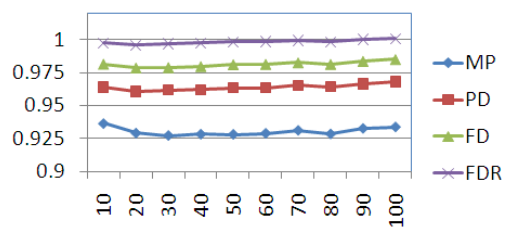
Fig. 13 shows the performances of the aggregation algorithms w.r.t k (the number of clusters) which varies from 10 to 100. The baseline algorithm is IA. No matter what number of clusters to perform, the four algorithms always beat the baseline in speed. With the increase of k, all the four algorithms (MP, PD, FD and FDR) tend to increase the quality (the FDR algorithm can even outperform the baseline). The reason is that larger k means more meta-points, which results in finer summarization of the individual points, thus achieves better quality. Correspondingly, the speedup decreases because on average each meta-point will represent less number of individual points.

5.2.6 *General Results*

Throughout all the above experiments, our clustering aggregation algorithms show much better performance than the non-aggregation baseline. MP gets as high as 94 times faster and about 93% approximate quality. PD gets as high as 67 times faster with 96% quality. MP gets as high as 54 times faster with 98% quality. FDR achieves as high as 20 times faster with almost the same quality (99.7%, sometimes even over 100%, which means even better quality) as the baseline.



(a) Time performance (Speedup)



(b) Quality performance (QR)

Figure 13: Performances of the algorithms w.r.t k, the number of clusters (X-axis). Y-axis denotes the evaluation metric. ($n = 10000, d = 6$)

5.3 Discussion

The cube aggregation algorithms work by aggregating from a single selected supporting cuboid and ignore other cuboids. An ensemble based aggregation which utilizes all candidate cuboids could be adopted to improve quality. When clustering quality is more important than cube construction speed, we can first perform temporary individual aggregations from each candidate supporting cuboid and then ensemble the individual aggregations to get a good final aggregation by consensus decision [15].

6. CONCLUSIONS

In this paper, we presented a framework for OLAP of images and proposed the Visual Cube to support efficient OLAP analysis. Our contribution is summarized as follows.

(1) We designed Visual Cube and proposed algorithms for Visual Cube construction. For the clustering structure measure, in addition to the intuitive algorithm of independent aggregation, we also proposed more efficient algorithms which are much faster while being able to achieve similar quality (in some cases even better quality).

(2) We proposed the idea of dynamic aggregation selection which can improve the performance of data cube computation.

(3) We introduced the *cell overlapping* issue in Visual Cube and OLAP. Overlapping has a great impact on the aggregation type of data cube measures, computation and OLAP operations. We proposed efficient solution and a new type of OLAP operation to support overlapping.

Some future work include: (1) OLAP based Image Retrieval; (2) Cell Level Top-k Query, given query cell, find top-k similar cells measured by the similarity of their images; (3) Visual frequent patterns mining in Visual Cube; and (4) Incremental Visual Cube, given new images, efficiently update the cube.

Acknowledgment

Research was sponsored in part by Kodak Inc., NSF grants IIS-09-05215, AFOSR MURI award FA9550-08-1-0265, and by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053 (NS-CTA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

7. REFERENCES

[1] G. G. Angelica. Applying OLAP Pre-Aggregation Techniques to Speed Up Query Processing in Raster-Image Databases. In *GI-DAYS*, 2007.

[2] C. K. Chui. The design and implementation of an olap system for sequence data analysis. In *IDAR '08: Proceedings of the 2nd SIGMOD PhD workshop on Innovative database research*, pages 1–6. ACM, 2008.

[3] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2):1–60, April 2008.

[4] Flickr. `http://www.flickr.com`.

[5] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Min. Knowl. Discov.*, 1(1):29–53, 1997.

[6] J. Han and M. Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, second edition, March 2006.

[7] F. M.-f. Jiang, J. Pei, and A. W.-c. Fu. Ix-cubes: iceberg cubes for data warehousing and olap on xml data. In *CIKM '07: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 905–908. ACM, 2007.

[8] X. Jin, S. Kim, J. Han, L. Kao, and Z. Yin. Gad: General activity detection for fast clustering on large data. In *2009 SIAM Int. Conf. on Data Mining (SDM'09)*, Sparks, NV, USA, 2009.

[9] H. Labs. Hunter lab color scale. *Insight on Color*, 8 9:1–15, 1996.

[10] J. Li, H. Zhou, and W. Wang. Gradual cube: Customize profile on mobile olap. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, pages 943–947, Washington, DC, USA, 2006. IEEE Computer Society.

[11] C. X. Lin, B. Ding, J. Han, F. Zhu, and B. Zhao. Text cube: Computing ir measures for multidimensional text database analysis. In *Proceedings of IEEE ICDM'08*, pages 905–910, 2008.

[12] S. Merugu and J. Ghosh. A distributed learning framework for heterogeneous data sources. In *KDD '05: Proceedings of the eleventh ACM SIGKDD*, pages 208–217, 2005.

[13] X. Mingjie. Experiments on Remote Sensing Image Cube and its OLAP. In *IEEE International Geoscience and Remote Sensing Symposium IGARSS*, 2004.

[14] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[15] W. Punch. Clustering ensembles: Models of consensus and weak partitions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(12):1866–1881, 2005.

[16] L. SP. Least squares quantization in pcm. Technical Report RR-5497, Bell Lab, September 1957.

[17] S. Wang, F. Jing, J. He, Q. Du, and L. Zhang. Igroup: presenting web image search results in semantic clusters. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 587–596, 2007.

[18] K. Wu, E. Otoo, and A. Shoshani. On the performance of bitmap indices for high cardinality attributes. In *VLDB '04: Proceedings of the Thirtieth international conference on Very large data bases*, pages 24–35. VLDB Endowment, 2004.

[19] O. R. Zaïane, J. Han, Z.-N. Li, S. H. Chee, and J. Y. Chiang. Multimediaminer: a system prototype for multimedia data mining. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 581–583. ACM, 1998.

[20] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An array-based algorithm for simultaneous multidimensional aggregates. *SIGMOD Rec.*, 26(2):159–170, 1997.