An Efficient Clustering Method for High-Dimensional Data Mining

Jae-Woo Chang and Yong-Ki Kim

Dept. of Computer Engineering
Research Center for Advanced LBS Technology
Chonbuk National University, Chonju, Chonbuk 561-756, South Korea
{jwchang,ykkim}@dblab.chonbuk.ac.kr

Abstract. Most clustering methods for data mining applications do not work efficiently when dealing with large, high-dimensional data. This is caused by so-called 'curse of dimensionality' and the limitation of available memory. In this paper, we propose an efficient clustering method for handling of large amounts of high-dimensional data. Our clustering method provides both an efficient cell creation and a cell insertion algorithm. To achieve good retrieval performance on clusters, we also propose a filtering-based index structure using an approximation technique. We compare the performance of our clustering method with the CLIQUE method. The experimental results show that our clustering method achieves better performance on cluster construction time and retrieval time.

1 Introduction

Data mining is concerned with extraction of information of interest from large amounts of data, i.e. rules, regularities, patterns, constraints. Data mining is a data analysis technique that has been developed from other research areas such as Machine Learning, Statistics, and Artificial Intelligent. However, data mining has three differences from the conventional analysis techniques. First, while the existing techniques are mostly applied to a static dataset, data mining is applied to a dynamic dataset with continuous insertions and deletions. Next, the existing techniques manage only errorless data, but data mining can manage data containing some errors. Finally, unlike the conventional techniques, data mining generally deals with large amounts of data.

The typical research topics in data mining are classification, clustering, association rule, and trend analysis, etc. Among them, one of the most important topics is clustering. The conventional clustering methods have a critical drawback that they are not suitable for handling large data sets containing millions of data units because the data set is restricted to be resident in a main memory. They do not work well for clustering high-dimensional data because their retrieval performance is generally degraded as the number of dimensions increases. In this paper, we propose an efficient clustering method for dealing with a large amount of high-dimensional data. Our clustering method provides an efficient cell creation algorithm, which makes cells by splitting each dimension into a set of partitions using a split index. It also provides a cell insertion algorithm to construct clusters of cells with more density than a given threshold as well as to insert the clusters into an index structure. By using an approximation technique, we also propose a new filtering-based index structure to achieve good retrieval performance on clusters.

The rest of this paper is organized as follows. The next section discusses related work on clustering methods. In Section 3, we propose an efficient clustering method to makes cells and insert them into our index structure. In Section 4, we analyze the performances of our clustering method. Finally, we draw our conclusion in Section 5.

2 Related Work

Clustering is the process of grouping data into classes or clusters, in such a way that objects within a cluster have high similarity to one another, but are very dissimilar to objects in other clusters [1]. In data mining applications, there have been several existing clustering methods, such as CLARA(Clustering LARge Applications) [2], CLARANS(Clustering Large Applications based on RANdomized Search) [3], BIRCH(Balanced Iterative Reducing and Clustering using Hierarchies) [4], DBSCAN(Density Based Spatial Clustering of Applications with Noise) [5], STING(STatistical INformation Grid) [6], and CLIQUE(CLustering In QUEst) [7]. In this section, we discuss a couple of the existing clustering methods appropriate for high dimensional data. We also examine their potential for clustering of large amounts of high dimensional data.

The first method is STING(STatistical INformation Grid)[6]. It is a method which relies on a hierarchical division of the data space into rectangular cells. Each cell is recursively partitioned into smaller cells. STING can be used to answer efficiently different kinds of region-oriented queries. The algorithm for answering such queries first determines all bottom-level cells relevant to the query, and constructs regions of those cells using statistical information. Then, the algorithm goes down the hierarchy by one level. However, when the number of bottom-level cells is very large, both the quality of cell approximations of clusters and the runtime for finding them deteriorate.

The second method is CLIQUE(CLustering In QUEst)[7]. It was proposed for high-dimensional data as a density-based clustering method. CLIQUE automatically finds subspaces(grids) with high-density clusters. CLIQUE produces identical results irrespective of the order in which input records are presented, and it does not presume any canonical distribution of input data. Input parameters are the size of the grid and a global density threshold for clusters. CLIQUE scales linearly with the number of input records, and has good scalability as the number of dimensions in the data.

3 An Efficient Clustering Method

Since the conventional clustering methods assume that a data set is resident in main memory, they are not efficient in handling large amounts of data. As the dimensionality of data is increased, the number of cells increases exponentially, thus causing the dramatic performance degradation. To remedy that effect, we propose an efficient clustering method for handling large amounts of high-dimensional data. Our clustering method uses a cell creation algorithm which makes cells by splitting each dimension into a set of partitions using a split index. It also uses a cell insertion algorithm, which constructs clusters of cells with more density than a given threshold, and stores the constructed cluster into the index structure. For fast retrieval, we propose a filtering-based index structure by applying an approximation technique to our clustering method. The figure 1 shows the overall architecture of our clustering method.

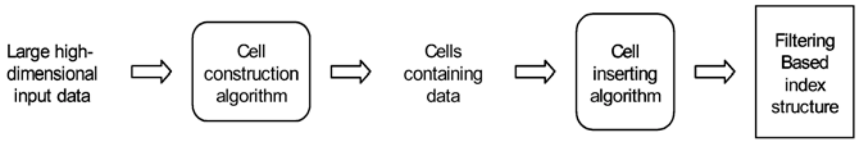
Large high-dimensional input data
Cell construction algorithm
Cells containing data
Cell inserting algorithm
Filtering Based index structure

Fig. 1. Overall architecture of our clustering method.

3.1 Cell Creation Algorithm

Our cell creation algorithm makes cells by splitting each dimension into a group of sections using a split index. Density based split index is used for creating split sections and is efficient for splitting multi-group data. Our cell creation algorithm first finds the optimal split section by repeatedly examining a value between the maximum and the minimum in each dimension. That is, it finds the optimal value while the difference between the maximum and the minimum is greater than one and the value of a split index after splitting is greater than the previous value. The split index value is calculated by Eq. (1) before splitting and Eq. (2) after splitting.

$$\mathrm{Split(S)} = 1 - \sum_{j=1}^{C} P_j^{\,2} \tag{1}$$

$$Split(S) = \frac{n_1}{n} Split(S_1) + \frac{n_2}{n} Split(S_2) \tag{2}$$

Using Eq. (1), we can determine the split index value for a data set S in three steps: i) divide S into C classes, ii) calculate the square value of the relative density of each class, and iii) subtract from one all the square values of the densities of C classes. Using Eq. (2), we compute a split index value for S after S is divided into S_1 and S_2. If the split index value is larger than the previous value before splitting, we actually divide S into S_1 and S_2. Otherwise, we stop splitting. Secondly, our cell creation algorithm creates cells being made by the optimal split sections for n-dimensional data. As a result, our cell creation algorithm creates fewer cells than the existing clustering methods using equivalent intervals. Figure 2 shows our cell creation algorithm. Here, the subprogram called 'Partition' is one that partitions input data sets according to attributes. The subprogram is omitted because it is very easy to construct it by slightly modifying the procedure 'Make_Cell'.

In Figure 3, we show an example of our cell creation algorithm. We show the process of splitting twenty records with two classes in two-dimensional data. The split index value for S before splitting is calculated as 1- $[(10/20)^2 + (10/20)^2] = 0.5$. A bold line represents a split index of twenty records in the X-axis. First, we calculate all the split index values for ten intervals. Secondly, we choose an interval with the maximum value among them. Finally, we regard the upper limit of the interval as a split axis. For example, for an interval between 0.3 and 0.4, the split index value is calculated as $(4/20)* [1- (3/4)^2 - (1/4)^2] + (16/20)*[1- (7/16)^2 - (9/16)^2] = 0.475$. For an interval between 0.4 and 0.5, the split index value is calculated as $(9/20)*[1- (6/11)^2 - (5/11)^2] + (11/20)*[1- (6/11)^2 - (5/11)^2] = 0.501$.

```
Procedure Make_Cell(attributes, input data set S)
Begin
  For each attribute in S do
    For each split_point in attribute do
      Compute $\frac{n_1}{n}(1-\sum P_j^2)+\frac{n_2}{n}(1-\sum P_j^2)$
      If $1-\sum P_j^2 > MAX\ [\frac{n_1}{n}(1-\sum P_j^2)+\frac{n_2}{n}(1-\sum P_j^2)]$ then return
      Else
        Split S into S1 and S2 by max split_point
        Partition(attribute, S1)
        Partition(attribute, S2)
      Endif
    End of for
  End of for
End.
```

Fig. 2. Cell creation algorithm.

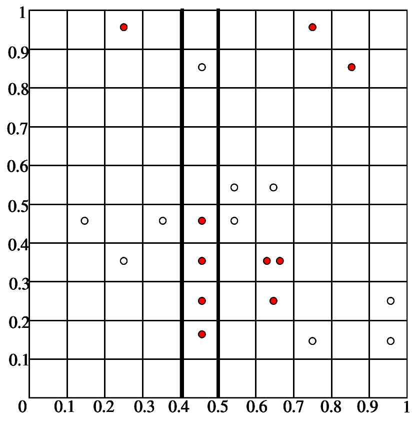

Fig. 3. Example of cell creation algorithm.

We determine the upper limit of the interval (=0.5) as the split axis, because the split index value after splitting is greater than the previous value. Thus, the X axis can be divided into two sections; the first one is from 0 and 0.5 and the second one is from 0.5 to 1.0. If a data set has n dimensions and the number of the initial split sections in each dimension is m, the conventional cell creation algorithms make m^n cells, but our cell creation algorithm makes only $K_1*K_2*...*K_n$ cells ($1 \leq K_1, K_2, .. ,K_n \leq m$).

3.2 Cell Insertion Algorithm

Using our cell creation algorithm, we obtain the cells created from the input data set. Figure 4 shows an insertion algorithm used to store the created cells. First, we construct clusters of cells with more density than a given cell threshold and store them into a cluster information file. In addition, we store all the sections with more density than a given section threshold, into an approximation information file.

```
Procedure Insert_Cell(cells)
Begin
  For each cells which is made form make cell do
    Compare the cell-threshold with cell density
    If cell_density >cell-threshold then
      Insert cell-information into cluster_info_file
    Compare section-threshold with section_density
    If secton_density > section-threshold then
      Approximation_info_file[volume] =1
    Else Approximation_info_file[volume] =0
  End of for
End.
```

Fig. 4. Cell insertion algorithm.

$$\textit{Section threshold} = \begin{cases} \lambda = \dfrac{NR \times F}{NI} \\ NI: \text{the number of input data} \\ NR: \text{the number of sections per dimension} \\ F: \text{minimum section frequency being regarded as '1'} \end{cases} \qquad (3)$$

$$\textit{Cell threshold}(\tau): \text{positive integer}$$

The insertion algorithm to store data is as follows. First, we calculate the frequency of a section in all dimensions whose frequency is greater than a given section threshold. Secondly, in an approximation information file, we set to '1' the corresponding bits to sections whose frequencies are greater than the threshold. We set other bits to '0' for the remainder sections. Thirdly, we calculate the frequency of data in a cell. Finally, we store cell id and cell frequency into the cluster information file for cells whose frequency is greater than a given cell threshold. The cell threshold and the section threshold are shown in Eq. (3).

3.3 Filtering-Based Index Scheme

In order to reduce the number of I/O accesses to a cluster information, it is possible to construct a new filtering-based index scheme using the approximation information

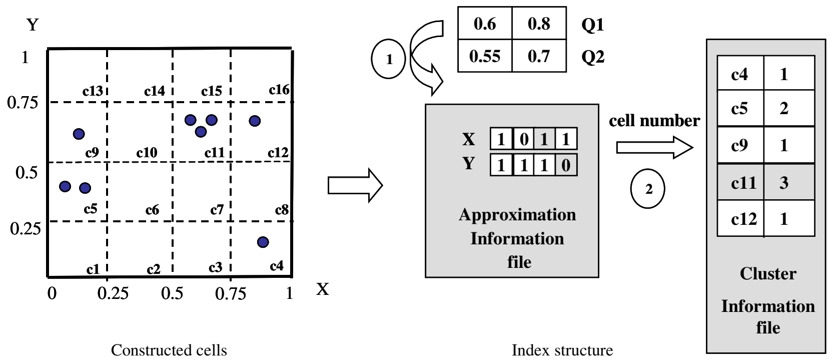

Fig. 5. Two-level filtering-based index scheme.

file. Figure 5 shows a two-level filter-based index scheme containing both the approximation information file and cluster information file.

Let assume that K clusters are created by our cell-based clustering method and the numbers of split sections in X axis and Y axis are m and n, respectively. The following equation, Eq.(4), shows the retrieval times (C) when the approximation information file is used and without the use of it. We assume that α is an average filtering ratio in the approximation information file. D is the number of dimensions of input data. P is the number of records per page. R is the average number of records in each dimension. When the approximation information file is used, the retrieval time decreases as α decreases. For high-dimension data, our two-level index scheme using the approximation information file is an efficient method because the K value increases exponentially in proportion to dimension D.

i) Retrieval time without the use of an approximation information file
$C = \lceil K/P \rceil / 2$ (Disk I/O accesses)

ii) Retrieval time with the use of an approximation information file

$$C = \lceil (D * R)/P \rceil * \alpha + (1-\alpha) \lceil K/P \rceil / 2 \text{ (Disk I/O accesses)} \tag{4}$$

When a query is entered, we first obtain sections to be examined in all the dimensions. If all the bits corresponding to the sections in the approximation information file are set ‘1’, we calculate a cell number and obtain its cell frequency by accessing the cluster information file. Otherwise, we can improve retrieval performance without accessing the approximation information file. Increase in dimensionality may cause high probability that a record of the approximation information file has zero in at least one dimension.

Figure 5 shows a procedure used to answer a user query in our two-level index structure when a cell threshold and a section threshold are 1, respectively. For a query Q1, we determine 0.6 in X axis as the third section and 0.8 in Y axis as the fourth section. In the approximation-information file, the value for the third section in X axis is '1' and the value for the 4-th section in Y axis is '0'. If there are one or more sections with '0' in the approximation-information file, a query is discarded without searching the corresponding cluster information file. So, Q1 is discarded in the first phase. For a query Q2, the value of 0.55 in X axis and the value of 0.7 in Y axis belong to the third section, respectively. In the approximation information file, the third bit for X axis and the third bit for Y axis have '1', so we can calculate a cell number and obtain its cell frequency by accessing the corresponding entry of the cluster information file. As a result, in case of Q2, we obtain the cell number of 11 and its frequency of 3 in the cluster information file.

4 Performance Analysis

For our performance analysis, we implemented our clustering method on Linux server with 650 MHz dual processors and 512 MB of main memory. We make use of one million 16-dimensional data created by Synthetic Data Generation Code for Classification in IBM Quest Data Mining Project [8]. A record in our experiment is composed of both numeric type attributes, like salary, commission, age, hvalue, hyears, loan, tax, interest, cyear, balance, and categorical type attributes, like level, zipcode, area, children, ctype, job. The factors of our performance analysis are cluster construction time, precision, and retrieval time. We compare our clustering method (CBCM) with the CLIQUE method, which is one of the most efficient conventional clustering method for handling high-dimensional data. For our experiment, we make use of three data sets, one with random distribution, one with standard normal distribution (variation=1), and one with normal distribution of variation 0.5. We also use 5 and 10 for the interval of numeric attributes. Table 1 shows methods used for performance comparison in our experiment.

Table 1. Methods used for performance comparison (MI:Maximal Interval).

Methods	Description
CBCM-5R	CBCM for data set with random distribution(MI = 5)
CLIQUE-5R	CLIQUE for data set with random distribution (MI=5)
CBCM-10R	CBCM for data set with random distribution (MI=10)
CLIQUE-10R	CLIQUE for data set with random distribution (MI=10)
CBCM-5SND	CBCM with standard normal distribution (MI=5)
CLIQUE-5SND	CLIQUE with standard normal distribution (MI=5)
CBCM-10SND	CBCM with standard normal distribution (MI=10)
CLIQUE-10SND	CLIQUE with standard normal distribution (MI=10)
CBCM-5ND(0.5)	CBCM with normal distribution of variation 0.5 (MI=5)
CLIQUE-5ND(0.5)	CLIQUE with normal dist. of variation 0.5 (MI=5)
CBCM-10ND(0.5)	CBCM with normal distribution of variation 0.5 (MI=10)
CLIQUE-10ND(0.5)	CLIQUE with normal dist. of variation 0.5 (MI=10)

Figure 6 shows the cluster construction time when the interval of numeric attributes equals 10. It is shown that the cluster construction time increases linearly in proportion to the amount of data. This result is applicable to large amounts of data. The experimental result shows that the CLIQUE requires about 700 seconds for one million items of data, while our CBCM needs only 100 seconds. Because our method creates smaller number of cells than the CLIQUE, our CBCM method leads to 85% decrease in cluster construction time. The experimental result with the maximal interval (MI)=5 is similar to that with MI=10.

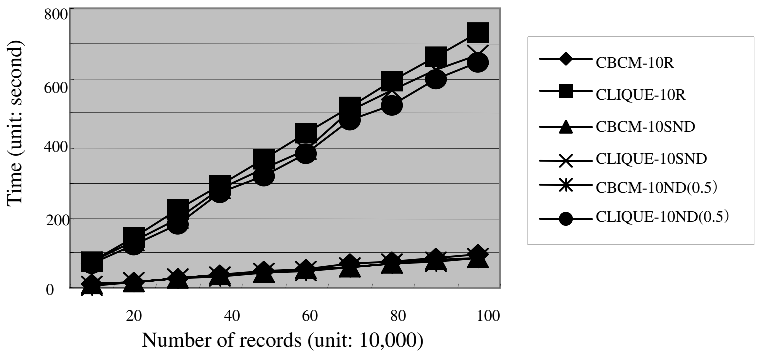

Fig. 6. Cluster Construction Time.

Figure 7 shows average retrieval time for a given user query after clusters were constructed. When the interval of numeric attributes equals 10, the CLIQUE needs about 17-32 seconds, while our CBCM needs about 2 seconds. When the interval equals 5, the CLIQUE and our CBCM need about 8-13 seconds and 1 second, respectively. It is shown that our CBCM is much better on retrieval performance than the CLIQUE. This is because our method creates a small number of cells by using our cell creation algorithm, and achieves good filtering effect by using the approximation information file. It is also shown that the CLIQUE and our CMCM require long retrieval time when using a data set with random distribution , compared with normal distribution of variation 0.5. This is because as the variation of a data set decreases, the number of clusters decreases, leading to better retrieval performance.

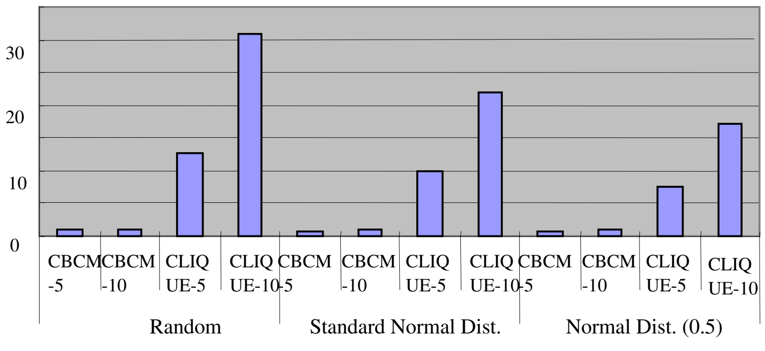

Fig. 7. Retrieval Time.

Figure 8 shows the precision of the CLIQUE and that of our CBCM, assuming that the section threshold is assumed to be 0. The result shows that the CLIQUE achieves about 95% precision when the interval equals 10, and it achieves about 92% precision when the interval equals 5. Meanwhile, our CBCM achieve over 90% precision when the interval of numeric attributes equals 10 while it achieves about 80% precision when the interval equals 5. This is because the precision decreases as the number of clusters constructed increases.

Because both retrieval time and precision have a trade-off, we estimate a measure used to combine retrieval time and precision. To do this, we define a system efficiency measure in Eq. (5). Here E_{MD} is the system efficiency of methods (MD) shown in Table 1 and W_p and W_t are the weight of precision and that of retrieval time, respectively. P_{MD} and T_{MD} are the precision and the retrieval time of the methods (MD). P_{MAX} and T_{MIN} are the maximum precision and the minimum retrieval time, respectively, for all methods.

$$E_{MD} = W_p \bullet \frac{P_{MD}}{P_{MAX}} + W_t \bullet \frac{1}{T_{MAX} / T_{MIN}} \tag{5}$$

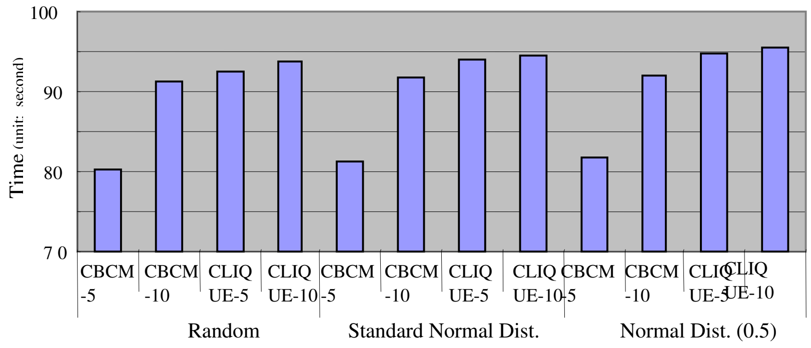

Fig. 8. Precision.

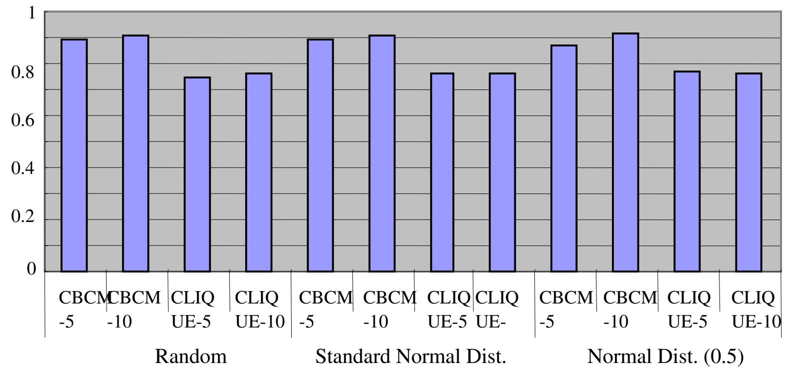

Fig. 9. System efficiency.

Figure 9 depicts the performance results of methods in terms of their system efficiency when the weight of precision are three times greater than that of retrieval time (W_p=0.75, W_t =0.25). It is shown from our performance results that our CBCM outperforms the CLIQUE with respect to the system efficiency, regardless of the data distribution of the data sets. Especially, the performance of our CBCM with MI=10 is the best.

5 Conclusion

The conventional clustering methods are not efficient for large, high-dimensional data. In order to overcome the difficulty, we proposed an efficient clustering method with two features. The first one allows us to create the small number of cells for large, high-dimensional data. To do this, we calculate a section of each dimension through split index and create cells according to the overlapped area of each fixed section. The second one allows us to apply an approximation technique to our clustering method for fast clustering. For this, we use a two-level index structure which consists of both an approximation information file and a cluster information file. For performance analysis, we compare our clustering method with the CLIQUE method. The performance analysis results show that our clustering method shows slightly lower precision, but it achieves good performance on retrieval time as well as cluster construction time. Finally, our clustering method shows a good performance on system efficiency which is a measure to combine both precision and retrieval time.

Acknowledgement

This research was supported by University IT Research Center Project.

References

1. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann (2000)
2. Ng R.T., Han J.: Efficient and Effective Clustering Methods for Spatial Data Mining. Proc. of Int. Conf. on Very Large Data Bases (1994) 144-155
3. Kaufman L., Rousseeuw P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. John Wiley & Sons (1990)
4. Zhang T., Ramakrishnan R., Linvy M.: BIRCH: An Efficient Data Clustering Method for Very Large Databases. Proc. of ACM Int. Conf. on Management of Data (1996) 103-114
5. Ester M., Kriegel H.-P., Sander J., Xu X.: A Density Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proc. of Int. Conf. on Knowledge Discovery and Data Mining (1996) 226-231
6. Wang W., Yang J., Muntz R.: STING: A Statistical Information Grid Approach to Spatial Data Mining. Proc. of Int. Conf. on Very Large Data Bases (1997) 186-195
7. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic Subspace Clustering of High Dimensional Data Mining Applications. Proc. of ACM Int. Conf. on Management of Data (1998) 94-105
8. http://www.almaden.ibm.com/cs/quest