IEICE TRANS. INF. & SYST., VOL.E94–D, NO.10 OCTOBER 2011

LETTER

ROCKET: A Robust Parallel Algorithm for Clustering Large-Scale Transaction Databases*

Woong-Kee LOH†, Yang-Sae MOON††, *and* Heejune AHN†††a), *Members*

SUMMARY We propose a robust and efficient algorithm called ROCKET for clustering large-scale transaction databases. ROCKET is a divisive hierarchical algorithm that makes the most of recent hardware architecture. ROCKET handles the cases with the small and the large number of similar transaction pairs separately and efficiently. Through experiments, we show that ROCKET achieves high-quality clustering with a dramatic performance improvement.

key words: *divisive hierarchical clustering, large-scale transaction databases, parallelization*

1. Introduction

A *transaction* is a set of one or more items [6]. The transaction database is composed of a number of transactions, each of which can have a different number of items. A typical example is a market basket database [4], [6]; a transaction is a set of products simultaneously purchased by a customer at a market. The issue in clustering the transaction database is creating good clusters (or groups) of similar (or highly correlated) transactions and then utilizing them in various applications. For example, the clustering result in a market basket database can be used to find groups of customers with similar purchase preferences and utilize them in improving sales performance.

The challenge in clustering transaction databases is that most of them are composed of a very large number of transactions. The previous clustering algorithms could deal with only small-sized databases or could not generate high-quality clusters [4], [6], [7]. Moreover, the algorithms were designed without consideration on the up-to-date hardware architecture. Instead of raising clock speed, recent CPUs are constructed to have multiple, simultaneously running cores, and a few such CPUs are equipped in a single server machine. Hence, we need an algorithm that fully harnesses the recent hardware technology to achieve high-quality clustering efficiently on the large-scale transaction databases.

In this paper, we propose a robust and efficient algorithm called ROCKET** for clustering the large-scale transaction databases. ROCKET is a divisive hierarchical algorithm using the notion of *link* in ROCK algorithm [4]. ROCKET is a parallel algorithm making the most of recent hardware architecture. ROCKET also separates the cases with the small and the large number of similar transaction pairs and handles each case efficiently. Through a series of experiments, we show that ROCKET achieves high-quality clustering with a dramatic performance improvement; it outperformed ROCK by up to 55.2 times. We expect that ROCKET will achieve higher improvement than previous algorithms with the release of CPUs with more cores and servers with more CPUs in the near future.

2. Related Work

ROCK [4] is a clustering algorithm for the objects with categorical attributes. The transaction is a kind of categorical objects. ROCK presented a new notion of *link* as a similarity measure for categorical objects. The link is obtained as follows. Given a threshold θ ($0 \le \theta \le 1$), ROCK computes a $sim(p_i, p_j)$ value for each object pair (p_i, p_j). The $sim()$ measure can be differently defined according to the target database and is defined as Eq. (1) for two transactions T_1 and T_2 [4]:

$$sim(T_1, T_2) = \frac{|T_1 \cap T_2|}{|T_1 \cup T_2|}, \quad (1)$$

where $|T|$ represents the number of items in T. Two objects (or transactions) are defined as *neighbors*, if the $sim()$ value between them is greater than or equal to θ. The link between two objects p_i and p_j is computed as follows:

$$link(p_i, p_j) = \left|\left\{p_k | sim(p_i, p_k) \ge \theta \text{ AND } sim(p_j, p_k) \ge \theta\right\}\right|. \quad (2)$$

That is, the link between two objects is the number of their common neighbors.

ROCK is an agglomerative hierarchical algorithm that merges the most similar cluster pairs one by one based on the links. Initially, for each object, ROCK forms a cluster

Manuscript received February 28, 2011.
Manuscript revised June 7, 2011.

†The author is with the Department of Multimedia, Sungkyul University, Korea.
††The author is with the Department of Computer Science, Kangwon National University, Korea
†††The author is with the Department of Control & Instrumentation Engineering, Seoul National University of Science and Technology, Korea.

*This work was supported by the Korea Research Foundation Grant funded by the Korean Government (MOEHRD, Basic Research Promotion Fund) (KRF-2008-331-D00487). This work was partially supported by the Korea Ministry of Knowledge Economy through the Strategic Technology Development Project (10031824).

a) E-mail: heejune@seoultech.ac.kr (Corresponding author)
DOI: 10.1587/transinf.E94.D.2048

**It stands for an algorithm faster than ROCK.

containing only the object. Then, ROCK computes *goodness* $g(C_i, C_j)$, which is a function of links, for each cluster pair (C_i, C_j) and then merges the cluster pair with the highest goodness. The $g()$ values incorporating the merged cluster are newly computed. Again, the cluster pair with the highest goodness is merged. This procedure is repeated until there is no more cluster pairs to merge or the pre-specified number k of clusters are obtained.

ROCK has a high time complexity of $O(n^2 + nm_m m_a + n^2 \log n)$ [4], where n is the number of objects, and m_m and m_a are the maximum and the average numbers of neighbors, respectively. Regardless of the target database, ROCK has the time complexity higher than $O(n^2)$, and thus it can hardly be applied for large-scale transaction databases. It was assumed in [4] that m_m is practically close to m_a and much smaller than n; however, there actually exist many databases with both m_m and m_a close to n. In this case, the time complexity of ROCK reaches as high as $O(n^3)$. In this paper, we call this kind of databases as *dense databases* and discuss how to handle them in Sect. 3.3.

Wang et al. [6] presented a new notion of a *large item* and proposed a clustering algorithm based on the notion. We call the algorithm as *LARGE* in this paper. As an experimental result using the mushroom database†, many of clusters created by LARGE contained both edible and poisonous mushrooms [6], which made the users skeptical about the clustering quality of the algorithm.

Yang et al. [7] proposed the CLOPE algorithm, which showed better clustering performance and quality than ROCK [4] and LARGE [6]. As an experimental result using the mushroom database, CLOPE obtained a perfect clustering result when setting its parameter *repulsion* r as $r \geq 3.0$ [7]. However, CLOPE can run only on a single thread by its nature; even when running on a most up-to-date server, it uses the server's capability only partially and hence cannot have an ultimate performance improvement.

Feng et al. [3] proposed a parallel clustering algorithm that runs on a cluster system composed of multiple PCs. Bohm et al. [1] proposed a density-based clustering algorithm that harnesses Graphics Processing Unit (GPU). This algorithm improved the performance by up to 15 times than the DBSCAN algorithm [2].

The algorithm by Feng et al. [3] deals with the objects that can be represented as points in a d-dimensional space. The objects consist of d items (real numbers), and the order and the difference between any two items are well-defined. Guha et al. [4] showed that the clustering quality dramatically falls off by applying the algorithms such as the one by Feng et al. [3] to transaction databases. The algorithm by Bohm et al. [1] and DBSCAN [2] primarily deals with the same objects as the algorithm by Feng et al. [3]. If the algorithms are managed to deal with transaction databases, their time and space complexity reaches as high as $O(n^2)$, where n is the number of transactions.

3. ROCKET Algorithm

3.1 Basic Architecture

ROCKET is given the same parameter θ as ROCK [4]. ROCKET computes a link value for each transaction pair using Eq. (2) in the same manner as ROCK. We discuss more on this procedure in Sect. 3.3. ROCKET uses the notion of *connectedness* defined as follows. Two transactions T_i and T_j are defined to be (*directly*) *connected*, if it holds that $link(T_i, T_j) > 0$. The two transactions are defined to be *indirectly connected*, if the following holds:

$$\exists\, T_l (0 \leq l < p) \text{ s.t. } link(T_l, T_{l+1}) > 0\,, \tag{3}$$

where $T_0 = T_i$ and $T_p = T_j$ $(p \geq 1)$.

ROCKET forms the initial clusters with indirectly connected transactions. Two transactions T_i and T_j are in the same cluster, if and only if they are indirectly connected. The clusters obtained at this stage are identical with those obtained by ROCK with $k = 1$. The problem with these clusters is that they may be very big and have very low quality depending on the target database. Actually, by clustering the mushroom database with $\theta \leq 0.5$, we got a single big cluster containing the entire database of mushrooms.

ROCKET divides the clusters in the direction of improving their quality. The divided clusters are recursively divided if it is needed to improve their quality. This recursion stops when there is no more quality improvement.

The cluster division method of ROCKET is based on the following observation. Figure 1 shows the occurrence counts of link values of transaction pairs in a bad cluster that should be divided (needs quality improvement). The figure was obtained in the middle of clustering the mushroom database with $\theta = 0.8$. In the figure, we can find that the incorrect links connecting two transactions that should not be in the same cluster have small link values.

ROCKET divides clusters by removing incorrect links, and each cluster is divided independently of the others. For a cluster, ROCKET computes the mean μ and the standard deviation σ of link values. Given a parameter s, it removes

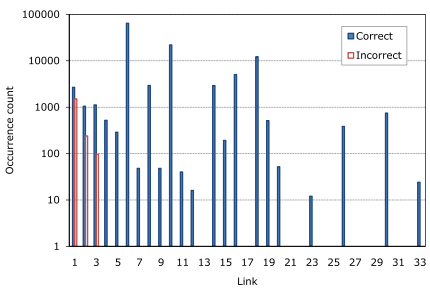

Fig. 1 Occurrence counts of link values in a bad cluster: incorrect links have small link values.

†It was obtained from UCI machine learning repository (http://archive.ics.uci.edu/ml/)

all the links l satisfying the following:

$$l \leq \mu - s\sigma \ . \tag{4}$$

Then, ROCKET forms (sub)clusters with indirectly connected transactions in the same manner as forming the initial clusters. This procedure is repeated until there are no links removed or the removal of links does not lead to a cluster division.

This cluster division method is simple, yet effective and fast. We found through experiments that s values around 1.0 were reasonable. The number of clusters k could be adjusted by changing s. Even with inattentive parameter θ values, we could achieve high-quality clustering by properly setting s values. We explain the reason that s was chosen as a parameter instead of k in the next section.

The cluster division method always generated a meaningful result using Eq. (4). Actually, the method performed not more than three iterations for any initial clusters in our experiments and returned high-quality clusters. The reason for the convergence is that, by applying the method repeatedly, the links satisfying Eq. (4) are removed very quickly, and there remain only the links with large link values. The convergence is dependent on the s value in Eq. (4); it should not be negative.

3.2 Parallelization

ROCKET has two methods for fully utilizing the up-to-date hardware architecture. The first is to have different clusters processed separately in different threads[†]. This method creates a critical section only for accessing the clusters queue shared by the threads. Since each cluster extracted from the queue can be processed independently of the others, as many clusters as threads can be processed in parallel, which results in a dramatic performance improvement. This is the reason that we designed ROCKET as a 'divisive' algorithm. We chose s as a parameter instead of k for the same reason. If k is used as a parameter, whenever we divide a cluster, we should select the most appropriate one from the candidate clusters. The next cluster to divide can only be selected after the previously selected cluster is divided and the number of current clusters is checked. That is, the clusters can only be processed sequentially.

The second method is to evenly distribute data into multiple threads when computing links for transaction pairs. As in ROCK [4], ROCKET creates a neighbor list NL_i for each transaction T_i. Then, for each pair of transactions T_j and T_l in NL_i, the algorithm increments the pair's occurrence count by 1. After processing NL_i for every transaction T_i, the count associated with each transaction pair (T_j, T_l) is its link $link(T_j, T_l)$ [4]. The second method divides the entire set of neighbor lists evenly into t sets, where t is the number of threads. Each set is processed separately in a different thread, and then the results from the threads are merged. In the merging phase, the links obtained from difference threads for the same transaction pair need only to be summed up.

3.3 Sparse and Dense Databases

As explained in the previous section, the neighbor list for every transaction is created to compute the link for every transaction pair. In this paper, we claim that the case with a small number of neighbors ($m_a \ll n$) and the case with a large number of neighbors ($m_a \approx n$) should be differently handled. The former is called *sparse databases* and the latter is called *dense databases*.

A straightforward method for creating neighbor lists is to compute a $sim()$ value using Eq. (1) for every combination of transactions T_i and T_j ($i \neq j$) and then to add T_j in NL_i and T_i in NL_j if it holds that $sim(T_i, T_j) \geq \theta$. However, in the case of sparse databases, i.e., when m_a is very small, since most of $sim()$ computation is needless, i.e., $sim() < \theta$, this method causes performance degradation. This method is effective in the case of dense databases, i.e., when m_a is close to n.

An efficient method for creating the neighbor lists in sparse databases is as follows. First, for each item e_i in every transaction, a list of transactions containing e_i is created. Then, for each pair of transactions T_j and T_l in the list, the occurrence count is incremented by 1. After processing all the transaction lists, the count associated with a transaction pair (T_j, T_l) is equal to the number of items commonly contained in two transactions, i.e., the denominator value in Eq. (1). The numerator value in Eq. (1) can be easily obtained using the property $|T_j \cup T_l| = |T_j| + |T_l| - |T_j \cap T_l|$. This method requires an execution time proportional to λ^2, where λ is the average size of transaction lists. In sparse databases, since it holds that $\lambda \ll n$, this method should have a short execution time. However, in dense databases, since λ is close to n, the execution time should be much longer. Actually, in our experiments, the straightforward method was more efficient in dense databases.

4. Evaluation

In this section, we evaluate the clustering quality and performance of ROCKET through a series of experiments. We use a mushroom database and an AOL database, which are typical examples of dense and sparse databases, respectively.

4.1 Mushroom Database

The mushroom database consists of 8124 mushroom records; each record contains categorical attributes on physical features such as color, odor, size, and shape. In addition, each record has an attribute indicating whether the mushroom is edible or poisonous. We transformed the database into a transaction database using a previously developed method [4], [7].

[†]For maximum hardware utilization, the number of threads t should be equal to the number of CPU cores p. If the CPU with Intel's Hyper-threading technology is used, t should be $2p$. If the server machine is equipped with m such CPUs, t should be $2pm$.

Table 1 Quality test using the mushroom database: ROCKET achieved 100% purity in every case.

θ	s	purity	k
0.9	1.00	100%	22
0.8	1.00	100%	25
0.7	0.85	100%	28
0.6	0.90	100%	42
0.5	0.75	100%	35

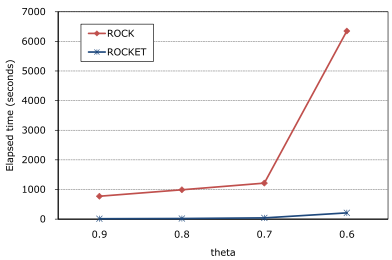

Fig. 2 Performance test using the mushroom database: ROCKET outperformed ROCK by up to 55.2 times.

We performed quality and performance tests using the mushroom database. In the quality test, purity values are computed for the clustering results by ROCKET obtained for various parameter values. The purity is defined as the number of mushrooms in pure clusters divided by the number of mushrooms in the entire database (= 8124), where a pure cluster contains either edible or poisonous mushrooms. The purity can have a value between 0% and 100%, and the higher, the better. Table 1 shows the result of the quality test; we could achieve 100% purity for various θ values by adjusting s appropriately. The fourth column (k) contains the number of final clusters.

In the performance test, the elapsed time of ROCKET and ROCK was compared for various parameter values. We set $k = 20$ for ROCK. Figure 2 shows the result of the performance test; ROCKET outperformed ROCK by up to 55.2 times and on the average by 40.9 times. The reason that the elapsed time for ROCK jumped up for $\theta = 0.6$ is that the number of neighbors dramatically increased.

4.2 AOL Database

The AOL database consists of 20M query records by about 650K users from March 1 through May 31, 2006. Each record contains a few attributes including user ID, query, and query time. We transformed the database into a transaction database using the method presented in [5].

We compared the elapsed time of ROCKET, ROCK, and CLOPE for various numbers of transactions. We set $\theta = 0.5$ and $s = 1.0$ for ROCKET, $k = 20$ for ROCK, and $r = 1.5$ for CLOPE. Figure 3 shows the result of the performance test; ROCKET outperformed ROCK by up to 10.0 times and on the average by 7.5 times, and it also outperformed CLOPE by up to 4.6 times and on the average by 4.1 times. Since the AOL database is a sparse database, the performance improvement ratio by ROCKET over ROCK was lower than that with the mushroom database. However, ROCKET fully utilizes the most up-to-date hardware architecture, and therefore we believe that the algorithm should achieve a higher performance improvement than the others with the advance in hardware technology.

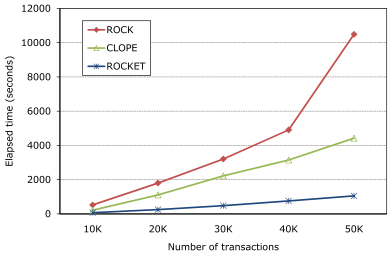

Fig. 3 Performance test using the AOL database: ROCKET outperformed ROCK by up to 10.0 times and CLOPE by up to 4.6 times.

References

[1] C. Bohm, R. Noll, C. Plant, and B. Wackersreuther, "Density-based clustering using graphics processors," Proc. Int'l Conf. on Information and Knowledge Management (CIKM), pp.661–670, Hong Kong, China, Nov. 2009.

[2] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," Proc. Int'l Conf. on Knowledge Discovery and Data Mining (KDD), pp.226–231, Portland, Oregon, USA, Aug. 1996.

[3] Z. Feng, B. Zhou, and J. Shen, "A parallel hierarchical clustering algorithm for PCs cluster system," Neurocomputing, vol.70, no.4-6, pp.809–818, Jan. 2007.

[4] S. Guha, R. Rastogi, and K. Shim, "ROCK: A robust clustering algorithm for categorical attributes," Information Systems, vol.25, no.5, pp.345–366, 2000.

[5] W.-K. Loh, Y.-S. Moon, and J.-G. Kang, "A data cleansing method for clustering large-scale transaction databases," IEICE Trans. Inf. & Syst., vol.E93-D, no.11, pp.3120–3123, Nov. 2010.

[6] K. Wang, C. Xu, and B. Liu, "Clustering transactions using large items," Proc. Int'l Conf. on Information and Knowledge Management (CIKM), pp.483–490, Kansas City, Missouri, USA, Nov. 1999.

[7] Y. Yang, X. Guan, and J. You, "CLOPE: A fast and effective clustering algorithm for transactional data," Proc. Int'l Conf. on Knowledge Discovery and Data Mining (KDD), pp.682–687, Edmonton, Alberta, Canada, July 2002.