A Large-Scale Performance Study of Cluster-Based High-Dimensional Indexing

Gylfi Pór Gudmundsson  
INRIA Rennes  
Campus de Beaulieu  
35042 Rennes, France  
Gylfi.Gudmundsson@inria.fr

Björn Pór Jónsson  
School of Computer Science  
Reykjavik University  
IS 101 Reykjavik, Iceland  
bjorn@ru.is

Laurent Amsaleg  
CNRS–IRISA  
Campus de Beaulieu  
35042 Rennes, France  
laurent.amsaleg@irisa.fr

ABSTRACT

High-dimensional clustering is used by some content-based image retrieval systems to partition the data into groups; the groups (clusters) are then indexed to accelerate processing of queries. Recently, the Cluster Pruning approach was proposed as a simple way to produce such clusters. While the original evaluation of the algorithm was performed within a text indexing context at a rather small scale, its simplicity motivated us to study its behavior in an image indexing context at a much larger scale. This paper summarizes the results of this study and shows that while the basic algorithm works fairly well, three extensions dramatically improve its performance and scalability, accelerating both query processing and the construction of clusters, making Cluster Pruning a promising basis for building large-scale systems that require a clustering algorithm.

Categories and Subject Descriptors

H.3.3 [Information Storage and Retrieval]: Clustering

General Terms

Algorithms, Performance

1. INTRODUCTION

Many approximate high-dimensional near neighbor search methods are based on some sort of segmentation of the data collection into groups of descriptors, which are stored together on disk. At query time, an index is then typically used to select the single nearest such group for searching. Several such methods are based on using a k-means approach for clustering the data. k-means, however, has problems at very large scale [8]. Chierichetti et al. [1] proposed a very simple algorithm, called Cluster Pruning, which uses the initial steps of the k-means algorithm to select a number of random cluster leaders and assign each descriptor to a single leader. At search time, the nearest b clusters are read and used to produce the approximate results. To improve result quality, they proposed some parameters affecting the size of clusters and the depth of the cluster index.

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.

VLS-MCMR’10, October 29, 2010, Firenze, Italy.
Copyright 2010 ACM 978-1-4503-0166-4/10/10 ...$10.00.

1.1 Scalability

While Cluster Pruning is efficient and effective, its performance was only studied using a small scale text collection. Its simplicity and performance is a strong motivation to study its behavior in an image indexing context at a larger scale, where secondary storage is needed.

State-of-the-art image applications typically use the SIFT descriptors [6] or variants thereof (e.g., [3, 4, 5]). These descriptors have two important properties that make them suitable for large-scale retrieval. First, they have been shown to scale very well with respect to result quality [5]. Second, each image is described by hundreds of descriptors, making approximate schemes fine for these applications. Because each image is described by hundreds of these high-dimensional descriptors, large-scale indexing and retrieval is absolutely necessary.

A major assumption made in the original design of Cluster Pruning is that CPU cost is dominant during the search. As a result of deciding to ignore disk cost, the optimal segmentation is to index a collection of n descriptors into \( \sqrt{n} \) clusters containing, on average, \( \sqrt{n} \) descriptors each; this minimizes the total CPU cost of the retrieval. While calculating euclidean distances is indeed CPU intensive, disk operations are also a significant source of cost. It is therefore necessary to study, for realistic workloads and data sets stored on disks, the optimal settings for the number of clusters and the resulting distribution of cluster sizes.

1.2 Contributions

In this paper, we study the performance of the Cluster Pruning algorithm in the context of a large-scale image copyright protection application [4, 5]. We study the effect of the various parameters of the Cluster Pruning algorithm, including index depth and cluster size, in this disk-based setting. Our results contradict some of the conclusions reached by Chierichetti et al. [1], due to the large scale of our experimental setup. While the basic algorithm still works fairly well, we propose three key changes which significantly improve its performance. First, a new parameter is needed to control cluster size on disk, to better balance IO and CPU costs. Second, a modification, which enables the use of the cluster index during the clustering phase, allows clustering the collection in a reasonable time. Third, by creating additional clusters and then reclustering the contents of the smallest clusters, cluster size distribution is improved which, in turn, improves search efficiency.

Note that, as mentioned above, there has been much recent research activity in the area of high-dimensional in-
indexing. As a result, there are other competing approaches, which have similar theoretical properties, but may be appropriate for different applications (e.g., see [7, 8, 11]). In this paper, we do not attempt a comparison of all these approaches, but focus instead on understanding the performance of one specific approach, the Cluster Pruning algorithm, for a particular workload setting. There is significant overlap between the ideas behind Cluster Pruning and the other approaches; Cluster Pruning can therefore be seen as a good representative for a whole family of algorithms where clustering is central. We thus believe that our analysis represents a very valuable contribution to the general understanding of disk-oriented cluster-based indexing.

The remainder of the paper is organized as follows. In Section 2 we review the copyright protection application we use in our work. We then review the Cluster Pruning algorithm in Section 3. In Section 4 we propose extensions to this algorithm for disk-based processing of large collections. In Section 5 we then run a detailed study of the impact of various parameters on performance. We discuss related work in Section 6, before concluding in Section 7.

2. IMAGE COPYRIGHT PROTECTION

We use as a case study the well known image copyright protection application. It is very different from the one studied by Chierichetti et al., where they used about 95,000 document descriptors with more than 400,000 dimensions. We describe this application and our experimental environment.

2.1 Image Collections and Queries

We use two collections of images. The first collection contains 30K high-quality news photos, which are very varied in content. The second collection, which includes the first collection, contains about 300K such photos.

Queries are intended to simulate image theft. The standard method for this purpose is to generate modified variants of images in the collection, for example using the StirMark software, and use those variants as queries. The goal is then to return the original image as a match, but no other images. For the purposes of our evaluation, 120 images were chosen at random from the collection, and modified with 26 different StirMark variants (resizing, cropping, compression, rotation, . . . ), resulting in 3,120 query images.

2.2 Descriptors and Query Model

Each image is described with many local descriptors, each describing a small portion of the image. We use the Eff² descriptors, which are a variant of SIFT [6], but perform significantly better for this application [5]. An Eff² descriptor has 72 dimensions, each stored in a byte. Additionally, each descriptor stores the identifier of the image it was extracted from, for a total of 76 bytes. The small collection has a total of 20,445,871 descriptors, while the large collection has 189,605,419 descriptors. The collections thus require 1.5GB and 13.4GB of disk storage, respectively. We believe that our collections are large enough for our conclusions to be quite general.

The descriptors from the images in the photo collections are stored in a large descriptor file, which is the input to the clustering process. When a query file is received, each of its q query descriptors is used in a k-nearest neighbor search: the closest cluster representative is first found, the contents of the cluster fetched in memory and distances finally computed to get the k neighbors. In this paper, we use k = 20, but the results are not very sensitive to that setting for large collections. Each neighbor vote for the image it was extracted from. These votes are aggregated over the image identifiers, and the images with the most votes are returned as an answer to the query.

2.3 Metrics

The cost of clustering and search is reported as wall-clock time. The search time reported corresponds to the average time spent to perform each of the 3,120 queries. Quality, on the other hand, is measured as follows. For each of the 3,120 query images, it is clear which image should be returned as a match. We consider an image a “correct match” when the correct image has at least twice as many votes as the image with the second most votes. The percentage of such correct matches is our baseline quality metric.

Note that the quality results in this study are lower than reported in many other studies, for three reasons. First, some of the StirMark variants are very difficult to find and even an exact sequential scan does not find all the correct matches. Second, a few of the selected images have near-duplicates in the collection, and therefore are never found as a correct match using our simple measure. Third, our criteria of having twice as many votes is very strict; it is possible to find a match with a relatively small number of votes by applying post-processing to the top images (e.g., see [4, 6]), but for simplicity we avoid such post-processing. The point of this study, however, is not to show that the descriptors are effective at image copyright protection—this is already known [4, 5, 6]. The main point is to investigate the performance of the Cluster Pruning algorithm, and this simple definition of a correct match suffices for that purpose.

3. THE CLUSTER PRUNING APPROACH

In this section, we first describe the basic Cluster Pruning approach, and then three parameters affecting its behavior.

3.1 The Cluster Pruning Algorithm

Assume a collection C = p₁, . . . , pₙ of n points in high-dimensional space. To form the clusters, a set of l = \sqrt{n} cluster leaders is first chosen randomly from C. Then, each pᵢ is compared to all l cluster leaders and assigned to its closest leader. Finally, once the clusters have been formed, a cluster representative is chosen, per cluster (obvious choices are the cluster leader itself, the centroid of the cluster, or the medoid of the cluster). At query time, the query point q is first compared to the set of l representatives to find the nearest one. Then, q is compared to all the points in that representative’s cluster, to find its k nearest neighbors.

The choice of l = \sqrt{n} clusters is made because the total number of euclidean distance calculations, which is l + nl, is minimized when l = \sqrt{n}. On average, each cluster contains \sqrt{n} points, resulting in a total of 2\sqrt{n} distance calculations. Assuming that the set of cluster representatives fits in memory, but not the descriptor collection, one disk read is required at search time. We now turn to the 3 parameters.

Extended Searches: The b Parameter

Sometimes, reading a single cluster may not yield results of satisfactory quality. In such cases, it is possible to read b clusters to answer each query; the basic algorithm corresponds to b = 1. The cost of retrieval then consists of b IOs and (1 + b)\sqrt{n}
distance calculations. Setting $b$ is difficult since the result quality is not predictable at run-time.

**Redundant Clustering: The $a$ Parameter.** Alternatively, it is possible to increase the quality of the results by assigning each data point to $a > 1$ clusters, and reading only $b = 1$ cluster at query time. Each cluster will then contain, on average, $a\sqrt{n}$ points, resulting in $(a+1)\sqrt{n}$ euclidean distance calculations, but only one IO. The clustering phase is always more costly with higher $a$ (the average cluster size is proportional to $a$). Furthermore, it is not possible to change the $a$ parameter once the clusters are formed, while the $b$ parameter can be dynamically modified at query time. Note it is possible to have both $a > 1$ and $b > 1$. Such settings will most likely result in several data points being read a times and are therefore not considered.

**Multi-Level Clustering: The $L$ Parameter.** For large collections, $\sqrt{n}$ is a large number, resulting in excessive CPU cost and potentially even significant IO cost. Chierichetti et al. propose to hierarchically cluster the set of representatives. Their parameter $L$ controls the number of levels in the hierarchy (the above description assumes $L = 1$).

Clustering with $L$ proceeds in a bottom-up fashion. First, $l = \frac{n^{\frac{1}{L}(L+1)}}{\sqrt{L}+1}$ leaders are chosen initially, resulting in $l$ clusters containing on average $n^{\frac{1}{L}(L+1)}$ descriptors. Cluster assignment then proceeds as before. Once the cluster representatives are chosen, however, they are considered as a collection of high-dimensional points, and clustered using $n^{\frac{(L-1)}{L}(L+1)}$ representatives. This process is repeated recursively, resulting in an $L$-tier index of cluster representatives, where each representative always represents, on average, $n^{\frac{1}{L}(L+1)}$ points at the next level. At query time, $(L+1)n^{\frac{1}{L}(L+1)}$ distances are calculated, while the number of IOs is at most $L$, assuming at least the top level fits in memory; most likely a few levels may fit in memory.

Note that the size of each cluster decreases rapidly as $L$ grows. This decreases CPU cost potentially at the expense of additional IOs.

### 3.2 Summary of Previous Results

While the bulk of the results reported by Chierichetti et al. [1] were obtained using a collection of about 95,000 descriptors with dimensionality of about 400,000, it is still instructive to recall their results.

Their goal was to determine the parameter settings that gave the best result quality in the shortest time span. First, they found cluster centroids to be the best representatives, followed by the cluster leaders. For that small collection, $L = 1$ gave the best results, followed closely by $L = 2$. Higher values of $L$ resulted in very poor results. They also found that for a memory-based setting using $a = 1$ worked best, as then $b$ could be varied to increase quality, while for a disk-based setting using $a = 5$ and $b = 1$ gave the best results. Our results, on the other hand, indicate that for large collections, using $L > 1$ and $a = 1$ is always preferred.

### 4. CLUSTER PRUNING EXTENSIONS

The main emphasis of the original algorithm was to minimize the CPU cost of queries. We now propose three new design choices that affect performance significantly.

#### 4.1 Choice of Cluster Representatives

We propose to use the cluster leaders as representatives, unlike Chierichetti et al. When cluster leaders are used, the bottom level of the cluster index is known before descriptors are assigned to clusters. Similarly, the upper levels of the cluster index are also known beforehand. As a result, the entire index exists before assigning any point to clusters. Subsequently assigning points to clusters can therefore benefit from that index, dramatically reducing clustering time. This optimization is not possible with the other choices of cluster representatives, as those are not known until the actual clusters have been created.

When using an index during cluster assignment, however, it is not clear that the most appropriate cluster is always found for all descriptors. To increase the likelihood of finding the best cluster for each descriptor, we always create the upper levels of the index using $a = 3$. While this setting does increase the index size, it can still easily fit in memory.

#### 4.2 Cluster Size Selection

The results reported in [10] indicated that cluster size is a key factor in the performance of cluster indexing, and that cluster size should be heavily influenced by the characteristics of the hard disk drive that descriptors reside on. In the original Cluster Pruning approach, however, there is a large difference in cluster sizes for $L = 1$ and $L = 2$, and both are independent of the IO granularity of the disk. While this behavior minimizes the CPU cost, increasing $L$ leads to very small descriptor clusters on disk, which under-utilize the IOs, and a correspondingly large index.

Instead of choosing $l = n^{\frac{L}{(L+1)}}$ leaders in the first step, we propose to give the desired average cluster size and then determine the number of leaders as follows:

$$l = \left\lfloor \frac{n}{\frac{\text{desired cluster size}}{\text{descriptor size}}} \right\rfloor$$  

Using this new number of cluster leaders, the clustering proceeds as before. When $L > 1$, each intermediate-level representative still represents $\sqrt{L}$ points at the next level.

By decoupling the size of the clusters from the choice of $L$, we gain two major benefits. First, larger clusters lead to a smaller index that may fit entirely in memory. Second, as each cluster is larger, fewer clusters may potentially be read. While CPU cost is sacrificed, the IO cost is reduced resulting in lower overall query processing cost.

#### 4.3 Balanced Size Distribution

In [10], it was shown that the largest natural clusters of a descriptor collection might be as large as 5-20% of the collection, while many clusters were very small. Small clusters still require an IO operation, while contributing little to the result quality. Large clusters result in both a more expensive IO operation and additional CPU cost. Both small and large clusters, therefore, reduce query processing performance. Furthermore, large clusters tend to get selected more often for processing than the average cluster, which impacts query processing even further.

In theory, the random leader selection process should generate equally sized clusters. In practice, however, several clusters are quite small and a few clusters are an order of magnitude larger. We propose a simple, yet surprisingly effective method to balance the size distribution. We intentionally choose $\times\%$ additional cluster representatives in the initial step of the algorithm. At the end of the cluster creation process we then eliminate the corresponding number of the smallest cluster leaders by reclustering their descrip-
Table 1: Impact of clustering and search performance (small coll., 128KB clusters, \( b = 5, a = 1 \)).

<table>
<thead>
<tr>
<th>Clustering</th>
<th>( L )</th>
<th>Search Time (sec) ( L = 1 )</th>
<th>Search Time (sec) ( L = 2 )</th>
<th>Correct Matches (%) ( L = 1 )</th>
<th>Correct Matches (%) ( L = 2 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.287.0</td>
<td>2.09</td>
<td>1.41</td>
<td>76.2</td>
<td>74.7</td>
</tr>
<tr>
<td>2</td>
<td>64.7</td>
<td>2.10</td>
<td>1.42</td>
<td>75.5</td>
<td>75.2</td>
</tr>
</tbody>
</table>

The last two columns show the search quality. Not surprisingly, the best quality is obtained through clustering and searching using \( L = 1 \). The most efficient combination is when \( L = 2 \) for clustering and search. The difference in quality is only 30 images, or less than 1% of the query set size. Given the tremendous efficiency gains, which will only become more important as the collections grow larger, the loss of quality is acceptable. We therefore only consider clustering that the best combination of clustering time, search performance and result quality is achieved using an average cluster size of 64KB or 128KB; we use 128KB in the remainder of our study.

Note that the original algorithm at \( L = 2 \) would have created about 74,000 clusters of about 16KB each; as our results show, these clusters would be far too small and many.

Table 2: Impact of average cluster size on clustering time (small coll., \( L = 2, a = 1 \)).

<table>
<thead>
<tr>
<th>( l ) (clusters)</th>
<th>Average Cluster Size (KB) (desc.)</th>
<th>Creation Time (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,964</td>
<td>512</td>
<td>23.3</td>
</tr>
<tr>
<td>5,928</td>
<td>256</td>
<td>38.2</td>
</tr>
<tr>
<td>11,859</td>
<td>128</td>
<td>66.0</td>
</tr>
<tr>
<td>23,719</td>
<td>64</td>
<td>97.8</td>
</tr>
<tr>
<td>47,438</td>
<td>32</td>
<td>146.0</td>
</tr>
</tbody>
</table>

Figure 1: Impact of average cluster size on search time (small coll., \( L = 2, a = 1 \)).

Figure 2: Impact of average cluster size on result quality (small coll., \( L = 2, a = 1 \)).
5.3 Impact of Redundancy

We now turn to the trade-off between the $a$ and $b$ parameters. Figure 3 shows the impact of $a$ and $b$ on search performance for two different memory settings. Consider first the results when the main memory allocation is 2GB. As expected, the results are identical for $a = b = 1$. Once $a > 1$, however, the performance becomes much worse than for corresponding settings of $b$. The primary reason for this difference is that when $a > 1$ clusters become much larger and therefore fewer can be cached in memory. Thus, each query must read most of its clusters from disk, while buffering performance is affected less by $b$.

To study the performance in a fair setting, we reduced the memory allocated to the operating system to 750MB and repeated the experiment. With this setting, both parameters are impacted by the buffer management performance, but varying $b$ is still more efficient. We believe there are two main reasons for this. First, even though few clusters fit in memory, clusters are still smaller and the buffer manager is therefore more likely to find them in memory. Second, since clusters are often larger than the IO granularity of the operating system, each “logical” IO may result in many “physical” IOs. This occurs more often with the larger clusters when using $a > 1$, which helps to explain the negative impact of $a$.

5.4 Impact of Cluster Size Distribution

Figure 4 shows the effect of our method for balancing clusters. The $x$-axis indicates how many additional clusters are created initially (percentage of the desired number of clusters). The $y$-axis shows the number of descriptors that fall into a given cluster size category; recall that the average size of clusters is 1,724 descriptors. As the figure shows, more than 10% of the data is initially ($X = 0$) either in very large or very small clusters, while only about 35% of the data is in the range from 1,000 to 2,000. As $X$ increases, the largest and smallest clusters shrink, and contain about 4% when $X = 100$, while 60% of the data then falls within the range from 1,000 to 2,000.

Figure 5 shows the impact of varying $X$ on clustering time, search time, and result quality, compared to $X = 0$. As expected, clustering time increases as $X$ is increased, due to the additional distance calculations, and nearly doubles when $X = 100$. Search time, on the other hand, decreases due to better size distribution of the clusters. Most importantly, however, result quality is only affected very slightly, as the number of correct matches only changes by ±10.

5.5 Impact of Scale

The previous experiments have studied the impact of various parameters at a moderate scale (although a collection of 20 million descriptors is, after all, quite large compared to the typical collections studied in the literature). We have concluded that for optimal performance, we should set $L = 2$ and $a = 1$, generate clusters with average size of 128KB, and use $b$ to improve result quality (optionally generating and then removing some extra clusters). We now apply these settings to a collection that is an order of magnitude larger. Note that in order to get a fair comparison of disk activity, we compared the search time of the large collection to the search time of the small collection with the 750MB configuration.

Since the collection is about 9.3 times larger, and cluster size is the same, there will be about 9.3 times as many clusters; as the depth of the index is the same, there will be about $\sqrt[9.3]{9.3}$ times more cluster representatives at each level. We therefore expect that the cluster creation will take about $9.3\sqrt[9.3]{9.3} \approx 28$ times more time, while the search should be affected much less. We also hope that the result quality will be largely unaffected.

Table 3 shows the results of the experiment. Clustering is about 36 times more time-consuming, which is close to the expectation. Searching is just over 2 times slower, mostly due to the additional cost of scanning the index, but potentially also due to a slightly worse cluster size distribution. Most importantly, however, the table shows that only 10 images are lost when going to the larger collection, which is a reduction of about 0.3%. As each descriptor is compared to

![Figure 3: Impact of $a$ and $b$ on search time (small coll., 128KB clusters, $L = 2$).](image)

![Figure 4: Data distribution for varying $X$ (small coll., 128KB clusters, $L = 2$, $a = 1$).](image)

![Figure 5: Relative performance for varying $X$ (small coll., 128KB cl., $L = 2$, $a = 1$).](image)

<table>
<thead>
<tr>
<th>Collection</th>
<th>Descriptors (millions)</th>
<th>Clustering Time (min)</th>
<th>Query Time (sec)</th>
<th>Matches (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>20.4</td>
<td>64.7</td>
<td>3.95</td>
<td>74.6</td>
</tr>
<tr>
<td>Large</td>
<td>189.6</td>
<td>2,344.7</td>
<td>8.82</td>
<td>74.3</td>
</tr>
<tr>
<td>Difference</td>
<td>≈9.3x</td>
<td>≈36x</td>
<td>≈2.2x</td>
<td>≈1x</td>
</tr>
</tbody>
</table>

Table 3: Comparison of the small and large collections (128KB clusters, $L = 2$, $b = 3$, $a = 1$).
only $3 \times 1,724 = 5,172$ descriptors on average, when $b = 3$, or about 0.003% of the collection, this is an excellent result.

5.6 Summary of Results

Several lessons can be drawn from the above experiments. First, multilevel clustering is necessary when indexing large collections. It allows for very efficient clustering when the index is created before assigning descriptors to clusters. Note that at even larger scales, when scanning the index becomes costly, incrementing the depth of the hierarchy may be considered. Second, partitioning the collection into I/O sized clusters is best for efficiency. This, together with a more balanced distribution of clusters sizes reduces the time spent on I/Os. Third, reading more than one cluster at search time yields the best result quality. It also absorbs the inaccuracies of assignments of points to clusters and compensates for the losses in precision due to the multiple levels of the hierarchy. Furthermore, compared to large clusters, it increases the chances of finding a cluster in memory, avoiding I/Os. All in all, these extensions help Cluster Pruning to scale very well to quite large data sets.

6. RELATED WORK

The seminal Video-Google approach to image retrieval uses k-means to group descriptors into visual words, which are then indexed using information retrieval techniques; in this case, the clusters are not used directly for query processing [11]. Philbin et al. [9] concluded, much like Chierichetti et al., that result quality is enhanced when varying the extent of the search and/or redundancy of the clustering.

Building on Video-Google, Nistér and Stewénius observed that the retrieval quality was increased when the visual vocabulary is significantly enlarged (to several millions) [7]. When $k$ is very large, however, standard $k$-means fails. They thus proposed a hierarchical $k$-means approach, which is quite similar to Cluster Pruning, but builds clusters top-down. Data points may be assigned to more than one leaf. Nistér and Stewénius do not study the various options discussed in the Cluster Pruning approach. They subsequently addressed the use of multiple (15–20) clusterings together to ensure quality, requiring one disk IO per cluster for each query descriptor [2].

Accelerating the clustering of the data collection in the Video-Google context is also the goal of Philbin et al. [8]. Their clustering process is flat, similar to standard $k$-means. They basically reduce the number of representatives each point must be compared to, boosting the assignment and trading-off speed for accuracy. They start by precomputing a large set of cluster representatives that get indexed into several randomized kd-trees. They assign a data point to its approximate closest representative by first probing each kd-tree with the point to cluster.

Overall, these methods [7, 8, 9, 11] have much in common with Cluster Pruning yet have quite specific properties. First, they never use the data in clusters, but rather the mapping between data points and cluster centers. Therefore, they are free to create poorly balanced clusters, and can rely on tf-idf schemes from information retrieval to compensate for differences in cluster cardinalities. Second, they also create a very large number of clusters since this, in turn, creates very sparse lists, as needed for efficient processing of inverted lists. Last, they are mostly main memory oriented.

7. CONCLUSIONS

Many content-based image retrieval systems and techniques rely on clustering to partition data, either for preprocessing or for data retrieval. Recently, the Cluster Pruning algorithm was proposed as a very simple, yet effective, approach for rapidly producing clusters of acceptable quality. Its simplicity and performance was a strong motivation to study its behavior in a large-scale image indexing context.

Building on Cluster Pruning, we have proposed three extensions which increase its performance at large scale. The first extension comes from the observation that disks can not be ignored and taking into account the IO granularity is a key factor to performance. The second extension comes from the observation that good search performance is obtained when clusters are better balanced. The third extension comes from the observation that, at scale, using an index to facilitate the assignment of data points to clusters is mandatory for performance.

Overall, we believe that, with our modifications, Cluster Pruning is a good basis for building large-scale systems that require a clustering algorithm. Not only is the algorithm fast, but it appears to produce clusters of acceptable quality, even at large scale.

Acknowledgement

We thank the project Quaero for its financial support.

8. REFERENCES