Image Cluster Compression using Partitioned Iterated Function Systems and efficient Inter-Image Similarity Features

Matthias Kramm
Technical University of Munich
Institute for Computer Science
Boltzmannstr. 3
D-85748 Garching
Email: kramm@in.tum.de

***Abstract*—When dealing with large scale image archive systems, efficient data compression is crucial for the economic storage of data. Currently, most image compression algorithms only work on a per-picture basis — however most image databases (both private and commercial) contain high redundancies between images, especially when a lot of images of the same objects, persons, locations, or made with the same camera, exist. In order to exploit those correlations, it's desirable to apply image compression not only to individual images, but also to groups of images, in order to gain better compression rates by exploiting inter-image redundancies.**

This paper proposes to employ a multi-image fractal Partitioned Iterated Function System (PIFS) for compressing image groups and exploiting correlations between images. In order to partition an image database into optimal groups to be compressed with this algorithm, a number of metrics are derived based on the normalized compression distance (NCD) of the PIFS algorithm. We compare a number of relational and hierarchical clustering algorithms based on the said metric. In particular, we show how a reasonable good approximation of optimal image clusters can be obtained by an approximation of the NCD and nCut clustering. While the results in this paper are primarily derived from PIFS, they can also be leveraged against other compression algorithms for image groups.

I. Introduction

Extending image compression to multiple images has not attracted much research so far. The only exceptions are the areas of hyperspectral compression [1]–[3] and, of course, video compression [4], which both handle the special case of compressing highly correlated images of exactly the same size.

Concerning generalized image group compression, we recently researched an algorithm which works by building a special eigenimage library for extracting principal component based similarities between images.

While the algorithm presented in [5] is quite fast, and manages to merge low-scale redundancy from multiple images, it fails to detect more global scale redundancies (in particular, similar image parts which are both translated and scaled), and also has the problem of becoming "saturated" quite fast (i.e., the more images in a group, the worse the additional compression rate of the individual images), which limits the size of possible image groups.

In this paper, we present a novel algorithm for image groups, which is based on PIFS compression [6], and thus manages to exploit several high-level redundancies, in particular scaled image parts.

Compression of image sequences using PIFS was done previously (in the context of video compression) in [7], [8]. However, in these papers, both the frames/images contributing to one compression group as well as the order of those images is predetermined by the video sequence. Furthermore, images need to be of the same size, which can't be assumed for most real-world image databases. Here, we specify a multi-image PIFS algorithm which works on images of arbitrary sizes, and also allows to cluster image databases into groups so that compression of each group is optimized.

The rest of this paper is organized as follows: We first derive the multi-image PIFS algorithm by generalizing the single-image PIFS algorithm. We also describe a way to optimize said algorithm using DFT lookup tables. Afterwards, we take on the problem of combining the "right" images into groups, by first describing efficient ways to compute a distance function between two images, and then, in the next session, comparing a number of clustering algorithms working on such a distance. The final algorithm is evaluated by compression runs over a photo database consisting of 3928 images.

II. The compression algorithm

PIFS algorithms work by adaptively splitting an image I into a number of non-overlapping rectangular "range" blocks $R_1 \dots R_n$ (using a quadtree algorithm with an error threshold ϵ_{max}), and then mapping each range block R onto a "domain" block D (with D being selected from a number of rectangular overlapping domain blocks $D_1, \dots, D_m$ from the same image) which is scaled to the dimensions of R by an affine transform, resulting in a block $\hat{D}$, and is henceforth processed by a contrast scaling c and a luminance shift l:

$$R_{xy} = c\hat{D}_{xy} + l \tag{1}$$

The contrast and luminance parameters can either be derived using a search operation from a number of discrete values $c_1, \ldots, c_N$ and $l_1, \ldots, l_M$:

$$c_{R,D}, l_{R,D} = \underset{c_i, l_j}{argmin} \sum_{x,y \in dim(R)} (c_i \hat{D}_{xy} + l_j - R_{xy})^2$$

They can also be calculated directly by linear regression:

$$c_{R,D} = \frac{|R| \sum \hat{D}_{xy} R_{xy} - \sum \hat{D}_{xy} \sum R_{xy}}{|R| \sum \hat{D}_{xy}^2 - (\sum \hat{D}_{xy})^2} \quad (2)$$

$$l_{R,D} = \frac{1}{|R|} (\sum R_{xy} - c_{R,D} \sum \hat{D}_{xy}) \quad (3)$$

with $\sum = \sum_{x,y \in dim(R)}$.

The quadratic error between a range and its domain block mapping is, for both cases:

$$\epsilon = \sum (c_{R,D} \hat{D}_{xy} + l_{R,D} - R_{xy})^2$$

which can also be written as

$$\epsilon = c_{R,D}^2 \sum \hat{D}_{xy}^2 + |R| l_{R,D}^2 + \sum R_{xy}^2 - 2 l_{R,D} \sum R_{xy} + \\ +2 c_{R,D} l_{R,D} \sum \hat{D}_{xy} - 2 c_{R,D} \sum \hat{D}_{xy} R_{xy}$$

(In [9], [10], the idea was brought forth to use the transform $R_{xy} = c(\hat{D}_{xy} - \sum \hat{D}_{xy}) + \sum R_{xy}$, so that only the contrast parameter c needs to be derived, which provides slightly better image quality if quantization is taken into account for the calculation of c. In our method, we however use the linear regression model, for simplicity.)

The domain block D to be mapped to the range block R needs to be searched in all available range blocks $\mathcal{D}_I$ from the image I, by minimizing:

$$D = \min_{D \in \mathcal{D}_I} \sum (c_{R,D} \hat{D}_{xy} + l_{R,D} - R_{xy})^2 \quad (4)$$

In the proposed multi-image compression method, equation (4) is now extended to a group of images $\mathcal{I}$:

$$D = \min_{\substack{D \in \mathcal{D}_I \\ I \in \mathcal{I}}} \sum (c_{R,D} \hat{D}_{xy} + l_{R,D} - R_{xy})^2 \quad (5)$$

Hence, domain blocks are collected from a number of images, and also images are allowed to cross-reference each other (see Fig. 1). Decompression works by crosswise recursion, in which all images are decompressed simultaneously (see Fig. 2).

In our implementation, we assume that domain blocks are always twice the size of range blocks, similar to the algorithm described in [11].

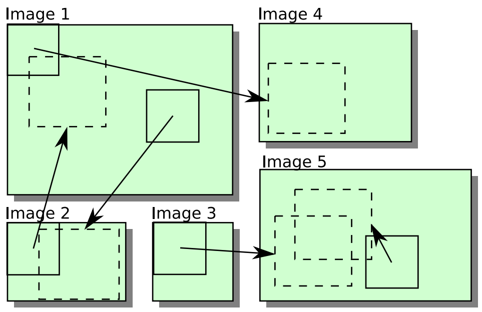

Fig. 1. *Cross references between PIFS compressed images of an image group*

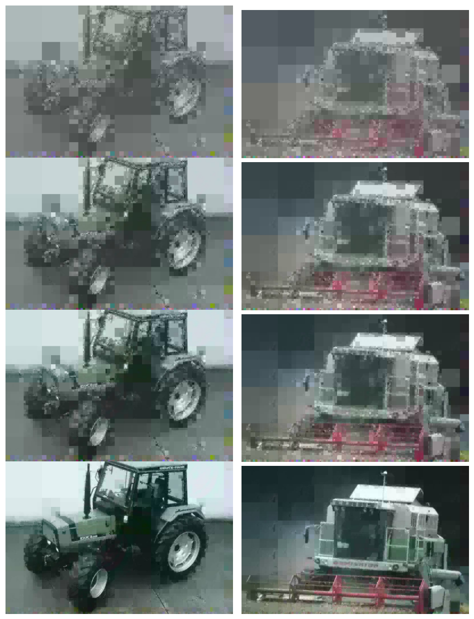

Fig. 2. *Crosswise recursive decompression of two images.*

Notice that e.g. in [12], methods which don't require any searching of domain blocks were devised — They do, however, base on the assumption that a given domain block is always most similar to it's immediate surroundings, a fact not extensible to multi-image compression.

Search of the domain blocks in our algorithm is performed by preprocessing a number of relevant parameters, in particular $\sum \hat{D}_{xy}$, $\sum \hat{D}_{xy}^2$ and $\sum R_{xy}$ and $\sum R_{xy}^2$, so that only

$$\sum R_{xy} \hat{D}_{xy} \quad (6)$$

needs to be calculated for each range and domain block combination.

The calculation of (6) as well as the preprocessing of $\sum \hat{D}_{xy}$, $\sum \hat{D}_{xy}^2$, $\sum R_{xy}$ and $\sum R_{xy}^2$ can be done very efficiently by using the Fast Fourier Transforms, analogous to the covariance method used in [13], which takes advantage of overlapping domain blocks:

$R_{xy} D_{xy}$ can be calculated for all domain blocks $D_i, \ldots, D_m$ simultaneously for a given R, by preprocessing

$\mathcal{F}(I_j)^C$ for all $I_j \in \{I_1, I_2, \ldots, I_n\}$ subsampled[1] by factor 2, and then filtering all those images using the range block ($A \cdot B$ denotes element-wise multiplication):

$$M = Cov(I_j, R) = \mathcal{F}^{-1}(\mathcal{F}(I_j)^C \cdot \mathcal{F}(R)) \tag{7}$$

In the array M, $M(u,v)$ will then contain $\sum R_{xy}\hat{D}_{xy}$ for the domain block $\hat{D}$ at the position $2u, 2v$ in the image to be compressed, so that the matching of a domain block against a range block can be done in 10 flops analogous to the single image variant presented in [13].

The algorithm hence uses one preprocessing loop and two nested loops over all images:

Algorithm 1 MULTI-IMAGE-PIFS$(I_1, I_2, \ldots, I_n)$

```
1:  for $I_j \in \{I_1, I_2, \ldots, I_n\}$ do
2:      Scale $I_j$ down by 2, store result in $S$
3:      Precalculate:
4:          $F_j = \mathcal{F}(S)^C$
5:          $r_{j,u,v} = \sum_{x<u,y<v} I_{j,x,y}$ for all $u, v$
6:          $r^2_{j,u,v} = \sum_{x<u,y<v} I^2_{j,x,y}$ for all $u, v$
7:          $d_{j,u,v} = \sum_{x<u,y<v} S_{x,y}$ for all $u, v$
8:          $d^2_{j,u,v} = \sum_{x<u,y<v} S^2_{x,y}$ for all $u, v$
9:  end for
10: for $I_j \in \{I_1, I_2, \ldots, I_n\}$ do
11:     for all range blocks $R \in I_j$ do
12:         Calculate $K = \mathcal{F}(R))$
13:         for $I_k \in \{I_1, I_2, \ldots, I_n\}$ do
14:             Calculate $M = \mathcal{F}^{-1}(F_k \cdot K)$
15:             for all domain blocks $D_{k,m} \in I_k$ do
16:                 Calculate $c, l$ from $M, r_j, r^2_j, d_k, d^2_k$
17:                 Quantize $c, l$
18:                 Calculate $\epsilon_{k,m}$ from $c, l, M, r_j, r^2_j, d_k, d^2_k$
19:             end for
20:         end for
21:         Find smallest $\epsilon$ from all $\epsilon_{k,m}$
22:         if $\epsilon > \epsilon_{\max}$ then
23:             Split range block $R$
24:         else
25:             Write out $k, m, c, l$
26:         end if
27:     end for
28: end for
```

III. Image similarity

Image databases typically consist of tens of thousands of images. The algorithm needs to compress all images in a group as a whole[2], and, more importantly, also needs to decompress the whole group in order to retrieve a single image.

Hence, it's desirable to split the input data into manageable clusters. Here, the opportunity presents itself to organize the clusters in a way that compression is optimized, i.e., that relevance is paid to the fact which images benefit most from each other if placed into the same cluster.

In order to partition the database in such a way, a metric specifying a kind of compression distance between to images need to be devised (so that the clustering algorithm will know which images are "similar" and should be placed in the same group).

Using the normalized compression distance (NCD) from [14], this can be expressed as

$$NCD_{I_1,I_2} = \frac{C_{I_1,I_2} - min(C_{I_1}, C_{I_2})}{max(C_{I_1}, C_{I_2})} \tag{8}$$

with $C_{I_1,\ldots,I_n}$ the compressed filesize of compressing images $I_1, I_2, \ldots, I_n$ together, and C_{I_k} the filesize of a compression run on just a single image.

This metric can both be interpreted as similarity between I_1, I_2 as well as the quality of a cluster formed by I_1, I_2. A lower value of NCD_{I_1,I_2} denotes that I_1 and I_2 have a more close resemblance.

It's important to notice that, in our case, the NCD is not necessarily a "true" metric (The PIFS compressor is not a "normal" compressor [14]). In particular, $NCD_{I,I} \neq 0$ if the PIFS algorithm considers only domain blocks larger than region blocks (As in our case[3]). This is due to the fact that the "second" image doesn't contain any additional information which improves the compression of the first image (The domain blocks don't differ from those already available from the first image).
This abnormal behaviour of the metric function disappears, however, once the images are at least slightly dissimilar (see Fig. 3), so this doesn't present a problem in practical applications.

We found that at least for some clustering algorithms, it's sometimes more efficient and produces better results if we work on a slightly simpler function, the function of preserved bytes:

$$b^+_{I_1,I_2,\ldots,I_n} = C_{I_1} + C_{I_2} + \ldots + C_{I_n} - C_{I_1,I_2,\ldots,I_n} \tag{9}$$

The function b^+ can also be applied to image groups of more than two images, and describes the number of bytes that were saved by combining the images $I_1, I_2, \ldots, I_n$ into a common cluster, which is also a more intuitive way of defining a similarity function. The higher the value of $b^+_{I_1,I_2,\ldots,I_n}$, the more resemblance between $I_1, I_2, \ldots, I_n$.

Since during clustering, a huge number of images need to be "compared", it's advisable to find faster approximations to

[1]We assume a fixed size for the Fourier transform, big enough to encompass all images $I_1, \ldots I_n$. When $\mathcal{F}$ is applied to an image or a block smaller than this size, it is zero-extended.

[2]Images can be added to a compressed file by allowing the new image to reference the already existing images, but not vice versa. Also, adding an image to a compressed file provides worse compression results compared to adding the image to the initial set. It's also slower, as all domain block information needs to be calculated again.

[3]It's possible for region blocks and domain blocks to be of the same size with the algorithm still converging, as long as the mappings between images are never circular. This can be accomplished by disallowing mappings from any image I_j to the images $I_j, I_{j+1}, \ldots, I_n$. Algorithm constructed using this kind of model bear a close resemblance to the motion compensation technique used in video compression.

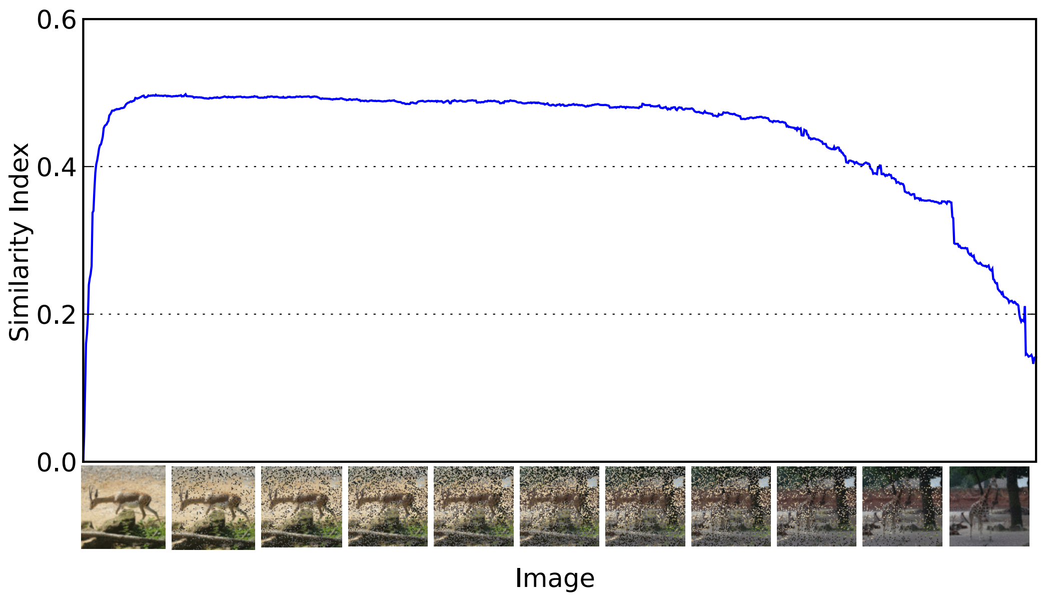

Fig. 3. *NCD Image similarity based on PIFS- an image is not similar to itself under the fractal NCD metric, if domain blocks are always larger than region blocks.*

the metrics (8) and (9). An obvious approach is to count the number of mappings spanning between images (i.e., where the domain block is in a different image than the range block), as opposed to mappings which stay in the image (domain block and range block are in the same image) — also see Fig. 5. It's also possible to, instead of counting the number of references, calculate the sum of $\epsilon_{I_j} - \epsilon_{I_1,\ldots,I_{j-1},I_{j+1},\ldots,I_n}$ for all inter-image references (with ϵ_{I_j} being the smallest error for domain blocks out of image I_j, and $\epsilon_{I_1,\ldots,I_{j-1},I_{j+1},\ldots,I_n}$ the smallest error for domain blocks out of all other images), a value which grows larger the more mapping error is reduced by introducing more images.

This type of metric has the advantage that we don't necessarily need to evaluate all range blocks, but can randomly pick a sample, and derive the metric only for that sample, therefore optimizing the speed the image comparison takes.

However, it's also a somewhat artificial approach, and doesn't relate too well to the PIFS compression properties — it doesn't take into account the extra bits we need in order to store (more) image-to-image references, and it's also hard to model the error threshold behaviour, where a new quadtree node is introduced every time a range block can't be sufficiently encoded on a specific scale, causing the output stream size to increase non-linearly.

Another option is to introduce an approximation to the NCD by subsampling the images in question by factor two, or even four, and then calculating the NCD on the subsampled images according to equation (8). This metric is a closer approximation to the NCD than the other two metrics (see Fig. 4), and is also the metric which we chose for subsequent experiments.

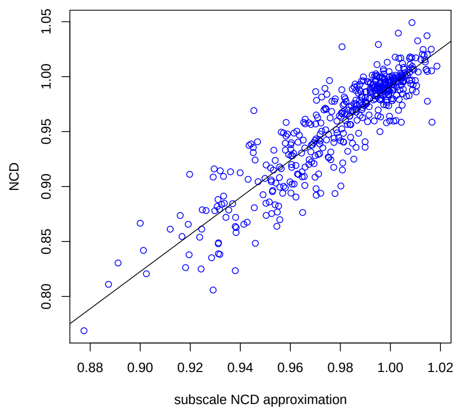

Fig. 4. *NCD compared with a NCD derived from images which were subsampled by factor two in both dimensions before compression. Each dot corresponds to an image pair, randomly selected from 128 example images.*

While the subsampling and following domain block searches can be performed quite efficiently (especially when using the precalculations for each image introduced in the last section), we still have to do a compression run on each image pair, which, especially for large databases, may take some time.

We hence also tried a different approach, and tested approximations of the "fractal similarity" using classical visual image similarity features, like *tf/idf* ratios of local Gabor filter and luminance histograms. These are features which can be precalculated for each individual image, and furthermore can

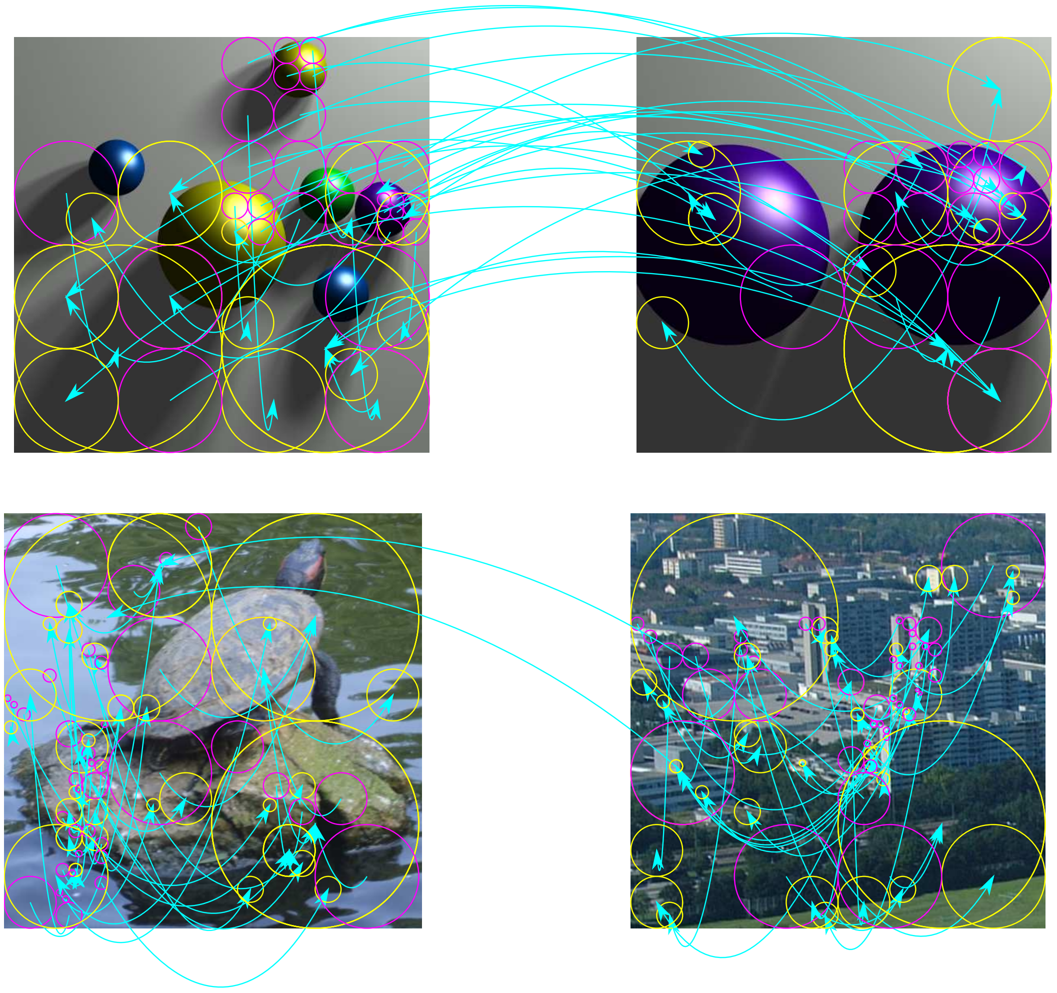

Fig. 5. *Two image pairs, which are considered more similar and less similar under the PIFS reference distance metric.*

also be stored in an inverted file, for faster lookup [15].

A software package exists [16], which can be used for extraction of these features, and for creating the inverted file database. We applied the algorithm to the grayscaled images only, so that the color histogram features of this package only measured the luminance components of our test data set.

IV. Clustering of Images

Using the metric b^+ from the previous section, one can create a weighted hypergraph out of the images (see Fig. 6), i.e. a hypergraph where each weight describes the number of bytes saved by combining the images the edge is adjacent to into a group.

If the only goal is to optimize compression rates (i.e., the final number of bytes the images use), we need to find the set of edges with the maximum weight which covers all vertices. In other words, we need to find a maximum matching for that hypergraph (See Fig. 7).
Maximum matching for (hyper-)graphs is an NP-hard problem, and, above all, would need **all** hyperweights ($2^n - 1$ values) in the given graph, so unfortunately, calculating the maximum matching is not a practical solution, and we have to find an approximation.

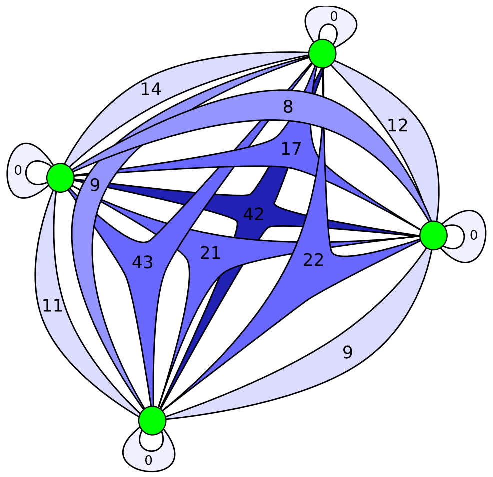

Fig. 6. *A b^+ weighted hypergraph. For every edge, the number of bytes saved by combining the corresponding images is depicted.*

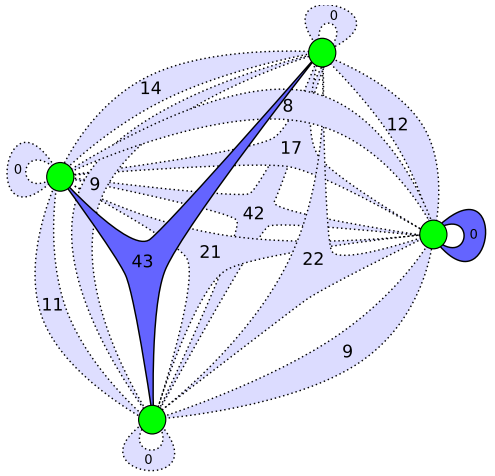

Fig. 7. *A maximum matching for the hypergraph from Fig. 6. By combining the upper, left and lower image into a group, and compressing the right image using single-image compression, the maximum number of bytes is saved.*

NCD metric			
Algorithm	Clusters	Compressed Size	Nodes per Cluster
SAHN	6	3792729	94 / 16 / 8 / 4 / 4 / 2
MST	6	3823392	123 / 1 / 1 / 1 / 1 / 1
k-Means	6	3852738	96 / 8 / 8 / 8 / 4 / 4
nCut	6	3864332	43 / 51 / 13 / 10 / 6 / 5
Random	6	3989745	24 / 24 / 24 / 22 / 20 / 14

Gabor filter metric			
Algorithm	Clusters	Compressed Size	Nodes per Cluster
MST	6	3880176	123 / 1 / 1 / 1 / 1 / 1
SAHN	6	3964413	69 / 45 / 10 / 2 / 1 / 1
nCut	6	3976120	28 / 25 / 21 / 19 / 18 / 17
k-Means	6	3987852	42 / 35 / 25 / 9 / 9 / 8
Random	6	3989745	24 / 24 / 24 / 22 / 20 / 14

TABLE I
COMPARISON OF DIFFERENT CLUSTERING ALGORITHMS ON A SAMPLE SET OF 128 IMAGES

Another problem is that since compression time grows quadratically with the number of images in a group, it's inefficient to compress image groups beyond a given size.

We found that by using clustering algorithms (a type of algorithm usually more common in the fields of data analysis and image segmentation), we can find approximations to the image grouping problem while using significantly less computing time.

We considered a number of different clustering algorithms, which all have different advantages and disadvantages, and which will described in the following.

- **MST clustering:** An algorithm which calculates the spanning tree from the distance metric, and then splits the tree into clusters by cutting off edges. [17] [18].
- **nCut clustering:** A hierarchical method which treats the complete data set as one big cluster, and then starts splitting the nodes into two halves until the desired number of clusters is reached (Splitting is done by optimizing the nCut metric [19]).
- **SAHN clustering:** Another hierarchical method, which in each step, combines a node (or cluster) and another node (or cluster), depending on which two nodes/clusters have the smallest distance to each other. Distances between clusters are evaluated using the sum over all distances between all nodes of both clusters, divided by the number of such distances [20].
- **Relational k-Means:** An extension of the “classical” k-Means of multidimensional data [21], which computes centers not by the arithmetic mean, but by finding a “median” node with the lowest mean distance to all other nodes [22].
- **Random clustering:** Distributes nodes between clusters arbitrarily. This algorithm was included for comparison purposes.

We did a comparison run of the aforementioned clustering algorithms on a small image database (128 images) using both the Gabor filter metric as well as the full NCD metric, in order to evaluate how much difference a more precise distance metric makes. The results are depicted in Table I.

For the tested data set, the MST and SAHN algorithms provide the best compression efficiency. MST unfortunately accomplishes that by creating a somewhat degenerated solution, which consists of removing the five images most dissimilar to the rest of the set, and creating a big cluster consisting of the remaining 123 images (MST therefore also creates the configuration which takes longest to compress). SAHN provides more evenly balanced clusters, which can be compressed faster. An even more evenly balanced configuration is created by nCut, however at the cost of slightly less compression efficiency.

It's worthwhile to note that the rough approximation to the NCD, using Gabor features, only results in a slight trade-off concerning compression efficiency, but has the advantage of greatly accelerating the speed with which the clustering is done — for 128 images, $\frac{1}{2}128 \cdot 128 + 128 = 8320$ compression runs otherwise need to be performed on single images and image pairs. For the feature based metric, on the other hand, only a few inverted file lookups are necessary [23].

While in the small sample set of 128 images, compressing an overlarge image group is still feasible, for larger image databases, care needs to be taken that clusters don't exceed beyond a maximum size. As the time needed for the compression is determined by the largest cluster in a given configuration (The algorithm is $O(n^2)$), care needs to be taken that the algorithm in question doesn't generate degenerate solutions (like the MST configuration from Table I) when processing larger data sets.

We accomplish this by recursively applying the clustering algorithm again to all image groups which are beyond a given threshold. For this evaluation, a maximal cluster size of 16 will henceforth be used. Furthermore, in order to prevent against excessive fragmenting, we iteratively combine pairs of groups which together are below that threshold into a common group. Some of the algorithms (like RACE) tend to create more balanced clusterings, and as such need fewer

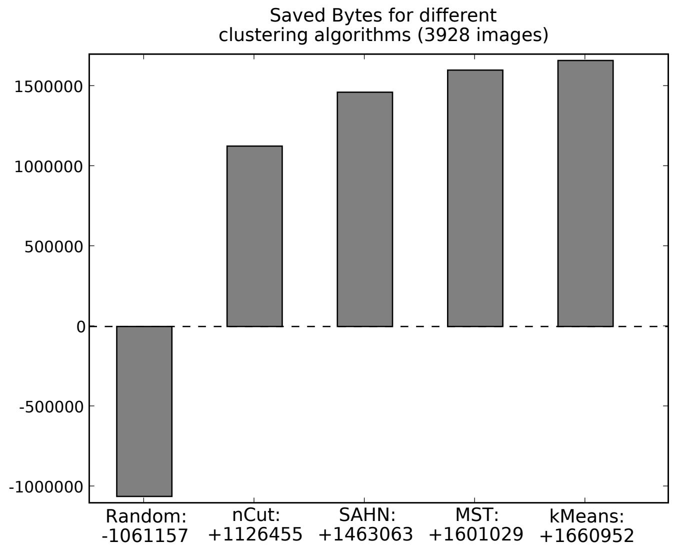

Fig. 8. *Compression result difference of our 3928 sample images against the filesize of single image compression, for different clustering algorithms.*

postprocessing than others (like MST or Greedy), which need several postprocessing iterations.

As some of the mentioned clustering algorithms are too expensive to apply on a large data set (e.g. nCut needs to solve a generalized Eigenvalue problem for a matrix of size $n \times n$ in order to cluster n images), we also fragment the data into chunks before the initial clustering. We used a chunk size of 256 in our experiments. This only applies to the MST, SAHN and nCut algorithms.

Using those algorithm improvements, we tested a data set of 3928 sample images[4], the results are depicted in Fig. 8.

We note that using an arbitrary clustering (Random), the compressions results are worse than with single-image compression. This happens because with more images in a given image group, also number of bits needed to encode the the inter-image differences grows. This puts further emphasis to the fact that in order to successfully apply multi-image compression, it's crucial to first cluster the images into well-matching groups.

We also note that technically superior algorithms (like nCut or SAHN) which can only be applied to subsets of the data are apparently less attractive than easier algorithms, like Relational k-Means, which can work on the full data set.

V. Concluding remarks

In this paper, we derived a new image cluster compression algorithm based on Fractal Partitioned Iterated Function Systems (PIFS) for multi-image compression, which is able to outperform its single-image variant considerably. We also presented methods for splitting image databases into manageable groups for compression with said algorithm. Using a feature-based metric, very large image databases can be partitioned into manageable clusters for being compressed with the multi-image PIFS algorithm. If the number of images is smaller and further compression efficiency is needed, the images can be be clustered using a more expensive metric, which clusters images using an approximation of the the Normalized Compression Distance (NCD) and produces better cluster configurations, at the cost of more computing time.

VI. Further research

The presented algorithm can easily be extended to irregular partitions, which in previous research have shown much better coding results [24].

We also would like to compare the compression rates of the algorithm with other (single-image) compression strategies, like JPEG2000 or JPEG XR. For these comparisons to be informative, the fractal coder also needs to employ a competitive empirical model for encoding the inter-image inter-block distances.

Furthermore, since apparently a connection exists between PIFS compressibility (and the simplified metrics) of two images and shared visual features of those images, an interesting research field is the approximation of visual image distinguishability using PIFS based NCD, similar to [25]. For this, the PIFS algorithm would optimally also support more fine-grained scaling and maybe even rotation.

We also plan to develop a number of other image cluster compression algorithms using different strategies, also extending into the field of lossless image compression.

Acknowledgment

This work is part of the IntegraTUM project, which was realised under partial funding by the German Research Foundation (DFG) from 2004 to 2009, and also funded (in the same amount) in the context of TUM's InnovaTUM program, with further substantial contributions by the permanent staff of the Technical University of Munich (TUM) and the Leibniz Supercomputing Centre (LRZ).

[4]We used images from our own image library, in particular a set consisting of agricultural images, containing both landscape, machinery and indoor photographs. The images are accessible at `http://mediatum2.ub.tum.de/node?id=11274&files=1`. The total size of the (uncompressed) images is 4.8 Gb.

References

[1] J. Saghri, A. Tescher, and J. Reagan, "Practical transform coding of multispectral imagery," 2005, pp. 32–43.

[2] J. Lee, "Optimized quadtree for karhunen-loeve transform in multispectral image coding," *Image Processing, IEEE Transactions on*, vol. 8, pp. 453–461, 1999.

[3] Q. Du and C.-I. Chang, "Linear mixture analysis-based compression for hyperspectral image analysis," *Geoscience and Remote Sensing, IEEE Transactions on*, vol. 42, pp. 875–891, 2004.

[4] L. Torres and E. Delp, "New trends in image and video compression," *Proceedings of the European Signal Processing Conference (EUSIPCO)*, pp. 5–8.

[5] M. Kramm, "Compression of image clusters using Karhunen Loeve transformations," in *Electronic Imaging, Human Vision*, vol. XII, no. 6492, 2007, pp. 101–106.

[6] M. Barnsley and A. Sloan, "A better way to compress images." *Byte*, vol. 13, no. 1, pp. 215–223, 1988.

[7] M. Lazar and L. Bruton, "Fractal block coding of digital video," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 4, no. 3, pp. 297–308, 1994.

[8] K. Barthel and T. Voye, "Three-dimensional fractal video coding," *Image Processing, 1995. Proceedings., International Conference on*, vol. 3, 1995.

[9] C. Tong and M. Wong, "Adaptive approximate nearest neighbor search for fractal image compression," *Image Processing, IEEE Transactions on*, vol. 11, no. 6, pp. 605–615, 2002.

[10] S. Furao and O. Hasegawa, "A fast and less loss fractal image coding method using simulated annealing," *Proceedings of Seventh Joint Conference on Information Sciences*, 2003.

[11] M. Nelson, *The data compression book*. M&T Books.

[12] S. Furao and O. Hasegawa, "A fast no search fractal image coding method," *Signal Processing: Image Communication*, vol. 19, no. 5, pp. 393–404, 2004.

[13] H. Hartenstein and D. Saupe, "Lossless acceleration of fractal image encoding via the fast Fourier transform," *Signal Processing: Image Communication*, vol. 16, no. 4, pp. 383–394, 2000.

[14] R. Cilibrasi and P. Vitani, "Clustering by Compression," *Information Theory, IEEE Transactions on*, vol. 51, no. 4, pp. 1523–1545, 2005.

[15] D. Squire, W. Müller, H. Müller, and T. Pun, "Content-based query of image databases: inspirations from text retrieval," *Pattern Recognition Letters*, vol. 21, no. 13-14, pp. 1193–1198, 2000.

[16] "GiFT - Gnu Image Finding Tool," http://www.gnu.org/software/gift/.

[17] Y. Xu, V. Olman, and D. Xu, "Minimum Spanning Trees for Gene Expression Data Clustering," *Genome Informatics*, vol. 12, pp. 24–33, 2001.

[18] A. Jain, M. Murty, and P. Flynn, "Data clustering: A review," *ACM Computing Surveys*, vol. 31, 1999.

[19] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[20] W. Day and H. Edelsbrunner, "Efficient algorithms for agglomerative hierarchical clustering methods," *Journal of Classification*, vol. 1, no. 1, pp. 7–24, 1984.

[21] D. Keim and A. Hinneburg, "Clustering techniques for large data sets — from the past to the future," *Conference on Knowledge Discovery in Data*, pp. 141–181, 1999.

[22] A. Hlaoui and S. Wang, "Median graph computation for graph clustering," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 10, no. 1, pp. 47–53, 2006.

[23] M. Rummukainen, J. Laaksonen, and M. Koskela, "An efficiency comparison of two content-based image retrieval systems, GiFT and PicSOM," *Proceedings of international conference on image and video retrieval (CIVR 2003), Urbana, IL, USA*, pp. 500–509, 2003.

[24] M. Ruhl, H. Hartenstein, and D. Saupe, "Adaptive partitionings for fractal image compression," *Proc. IEEE Int. Conf. Image Processing*, vol. 3, pp. 310–313, 1997.

[25] N. Tran, "The normalized compression distance and image distinguishability," *Proceedings of SPIE*, vol. 6492, p. 64921D, 2007.