Towards Automatic Detection of CBIRs Configuration

Christian Vilsmaier, Rolf Karp, Mario Döller,
Harald Kosch, and Lionel Brunie

University of Passau, Chair of Distributed and Multimedia
Information Sytems, Innstrasse 43, 94032 Passau, Germany
and
INSA de Lyon, LIRIS, Campus de la Doua, Batiment Blaise Pascal,
20 Avenue Albert Einstein, 69621 Villeurbanne Cedex, France
{Christain.Vilsmaier,Lionel.Brunie}@insa-lyon.fr,
{Harald.Kosch,Mario.Doeller}@uni-passau.de,
Rolf.Karp@googlemail.com

Abstract. Many *Content Based Image Retrieval systems* (CBIRs) have been invented in the last decade. The general mechanism of the search process is very similar for each of these CBIRs, and the calculation of rankings is determined by the comparison of features (low-, mid-, high-level). Nevertheless, all things being equal, the respective realization leads to different results. Knowledge about the internal configuration (used features, weights and metrics) of these systems would be beneficial in many usage scenarios (e.g., by using a query image content sensitive query forwarding strategy or improved result ranking strategies in meta search engines). In this context, the paper presents an approach that supports an automatic detection of the configuration of CBIR systems. We demonstrate that the problem can be partly traced back to an optimization problem and tested several optimization algorithms. The approach has been evaluated based on the ImageCLEF test set and shows good results.

Keywords: CBIRs configuration, Image Database, Low-level Feature detection.

1 Introduction and Background

Due to the digitalization and miniaturization of cameras as well as their integration into mobile phones, the amount of digitally available images has increased tremendously in the last decade. In order to make those images searchable based on content, CBIR has received a lot of attention in the last several years. The theoretical background which was developed during these years was exhaustively explored in [2] and [11] and various systems implementing CBIR were examined in [14] and [7]. A CBIR system (CBIRs) allows searching of images based on their extracted features (low-, mid-, high-level, see [15] for details) instead of textual descriptions. The retrieval process matches the extracted features of the stored images to those of a query image, thereby calculating its score and rank which subsequently results in a list of best matches. The general mechanism of

K. Schoeffmann et al. (Eds.): MMM 2012, LNCS 7131, pp. 334–345, 2012.

the search process is very similar for all of these CBIR systems. Nevertheless, they use different combinations of features, feature metrics and feature weights so that their search behavior and results differ.

In the context of meta-search engines [12] that provide access to multiple heterogeneous retrieval systems, the knowledge of their internal configuration would be beneficial. For instance such knowledge could allow the meta-search engines to establish an image content sensitive CBIR selection. Thus, CBIR systems that would not be of additional value to the query response can be ignored altogether. Furthermore, the gathered information can be used to improve the result aggregation process of the different retrieved result sets. Imagine examples of meta-search engines, for instance, in the domain of art galleries that provide search facilities of their works of art.

Related to this, the paper proposes a novel approach for an automatic detection of the configuration of CBIR systems. The detection process is based upon the analysis of a small set of test queries that are executed on the CBIR system in question. The analysis uses an optimization algorithm and filter strategies in order to identify the best feature/weight combination.

The remainder of this paper is organized as follows: In section 2 related work is discussed. In section 3 and 4 the assumptions and the developed approach are explained. In Section 5 the methods of evaluation as well as their results are analyzed. Finally in section 6 results and future work are discussed.

2 Related Work

The search in image repositories is a very active research field and many retrieval techniques and frameworks have been proposed in the past (see exhaustive surveys [14,7]). In this context, several articles focused on the qualitative evaluation of those techniques. For instance, in [4] the authors proposed an objective method for evaluating image content by means of visual content words as basis vectors for similarity calculations. Another important initiative in this domain is the ImageCLEF benchmark [10], which provides an annotated image test set.

Furthermore, in the literature detailed analyses can be found that investigate the individual features [15] and searches for correlations among them. Consequently, recommendations are given as to which features perform well for certain types of data [13].

Very little work related to the topic of our paper is available. For instance, the implementation of [1] presented a peer to peer approach for a self-organized image retrieval network. However, although the feasibility of an image retrieval network has been shown, little attention has been paid to identifying the configuration of the associated CBIR systems. The authors in [9] describe an algorithm for obtaining knowledge about the importance of features by analyzing user log files of the VIPER system. In their approach, features which are frequently present in images marked as positive by users receive a higher weighting. Moreover, their technique was used to improve retrieval quality due to a novel relevance feedback technique. However, their work relies on access to query logs and the internals of the system, which are not available in our case. It should

also be noted that our proposed approach does not aim at evaluating the quality of a CBIRs but at automatically detecting the used configuration.

3 Methodology

Current CBIR systems primarily use (low-level) features to compare query images to images that are stored in their system. Formula 1 shows the internal representation of an image i in a CBIRs α:

$$rep_\alpha(i) = \{f_\mathrm{j}(i) | j \in \{1..|F_\alpha|\}\} \tag{1}$$

The image i is represented as a set of feature vectors $f_\mathrm{j}()$, which are extracted from i. These features of the feature set F_α are digital representations such as color, edge or shape characteristics. Mixtures of such characteristics can also be represented in a feature. The used set of features may be different for every CBIR system. Even two retrieval systems which are using an equal set of features could reply differently to the same query as they might assign different weights or distance functions to their features. Formula 2 formalizes the *config*uration of CBIRs_α as a 3-tuple. This tuple consists of a set of features F_α, a set of feature metrics Δ_α and a set of feature weights W_α.

$$config(CBIRs_\alpha) = (W_\alpha, \Delta_\alpha, F_\alpha) \tag{2}$$

The calculation of CBIRs α's score respective to the query image q and a stored image i' uses the features, weights and metrics of this configuration. It is defined as follows: Let $score_{\alpha,\mathrm{i'}}(q)$ be the score of α respective to query image q and a stored image i' and let $\delta_{\alpha\mathrm{j}} \in \Delta_\alpha$ be the distance function of feature $f_\mathrm{j}()$. Furthermore, let $w_{\alpha\mathrm{j}} \in W_\alpha$ be the weight assigned by the system α to feature $f_\mathrm{j}()$. The score of image i' regarding the query image q is then calculated as follows:

$$score_{\alpha,\mathrm{q}}(i') = \sum_{j=1}^{|F_\alpha|} w_{\alpha\mathrm{j}} \delta_{\alpha\mathrm{j}}(f_\mathrm{j}(i'), f_\mathrm{j}(q)) \tag{3}$$

4 Approach Outline

As illustrated in section 1, our aim is an automatic detection of the configuration of a CBIRs. In this context the internal representation and the supported classification cases are presented in subsection 4.1. The overall process is then described in subsection 4.2 whereas the details of the algorithm are shown in subsection 4.3.

4.1 Feature Distribution

The basis of our approach is the exploitation of a rich set of implemented and well known features and feature metrics. However, for reasons of simplification

the description of the implementation explained in this article focuses on the features. Nevertheless, a metrics detection mechanism has also been integrated in our implementation and can be used by assigning multiple metrics to a feature. The internal representation of our system is analogous to that of the CBIRs introduced in section 3, methodology. Formula 4 shows the internal representation Rep_ω of an image i:

$$Rep_\omega(i) = \{f_\mathrm{j}(i)|j \in \{1..|F_\omega|\}\} \tag{4}$$

Since knowing all features, especially the proprietary ones that could be used in a CBIRs, is not feasible different classification cases have to be considered. Five such cases are possible (see also figure 1):

1) $F_\alpha = F_\omega$: In this case our representation is aware of exactly the same features the CBIRs is using. Here, only the weights for the features have to be found.
2) $F_\alpha \subset F_\omega$: In this case the features which are not used by α have to be identified and the weights for the remaining features have to be found.
3) $F_\alpha \supset F_\omega$: In this case the CBIR system in question uses features that are not present in our internal representation. The current focus of our implementation is the detection of this case in order to avoid false positives. Future research will consider feature classes (e.g. by statistical analysis according to the correlation of specific features [3]) by trying to identify which feature class has been used by the CBIRs.
4) $F_\omega \cap F_\alpha \neq \emptyset$: Not included in this case are constellations that are included in cases 1, 2 or 3. Similar to case 3, the current focus of our implementation is the detection of this case.
5) $F_\omega \cap F_\alpha = \emptyset$: In this case, as in cases 3 and 4, the current focus of our implementation is the detection of this constellation.

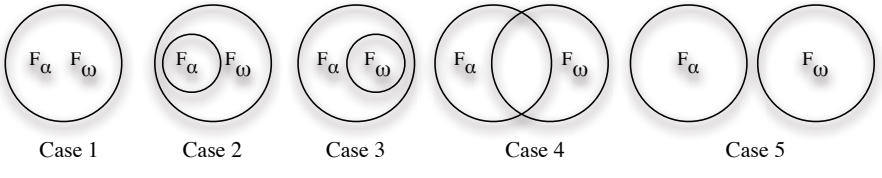

Fig. 1. Distinct cases for the detection of the configurations

4.2 Overall Process

The detection process of our algorithm starts with an enrollment of the CBIR in question.

Then a set of test images, from now on referred to as image test set (ITS), is used for querying the CBIRs (see figure 2). The selection of these images is arbitrary. The necessary size of ITS has been experimentally evaluated. In our tests setting the amount of test images at five showed a reasonable balance between processing speed and the accuracy of our detection approach.

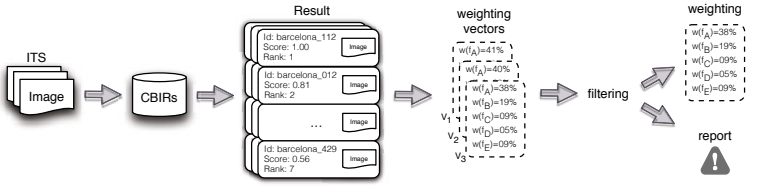

Fig. 2. Approach Outline

As shown in figure 2, for every single query image of the ITS the CBIRs responds with a ranked list of result images and associated scores.

Depending on the search paradigm used the length of those lists may vary. But as long as the returned number of results exceeds five items per list, for at least three returned result lists, those different paradigms do not influence the algorithms performance. The implemented paradigms of the CBIRs can therefore be left out of consideration. For the same reason false positives and negatives do not have to be considered. This is because of the approaches' exclusive interest in the detection of the configuration of the CBIRs. The approaches aim is not to judge the quality of the CBIR, but to detect its configuration.

Using the ranked lists received, we calculate the feature weighting vectors that generate self-computed scores very similar to the result scores. If no score is available, this calculation can also be made by the mere comparison of the ranks, which also omits a necessary normalization step. As this results in a poorer performance of the approach the scoring was preferred. This calculation is achieved by minimizing an objective function which evaluates possible weight vectors for the feature set by comparing our self-calculated scores to the result scores of the CBIR system. This way, the problem can be traced back to a vector optimization problem - an optimization algorithm can be used to find a good weight vector, preferably one very similar to the one the CBIRs uses internally. This optimization is performed for every query to be able to statistically evaluate the calculated optimal weight vectors. Confidence rating values are assigned to every feature according to different criteria, for example a weights' standard deviation across the multiple weight vectors. This rating is used to filter irrelevant features. Once there are no more features to filter, the feature weight configuration of the CBIRs is calculated using the arithmetical average.

4.3 Approach Details

Algorithm 1 presents the cornerstones of the described approach. The algorithm's goal is to find optimal weight values for every feature, in case every feature is known in our representation. Conversely, it's desirable that the algorithm reports if the analyzed CBIR system uses unknown features.

The algorithm is initialized with all features known to our internal representation (line 2). This set of possible candidates is then successively thinned out by removing irrelevant features (line 3). The *while*-loop in line 3 is repeated until no more features can be discarded due to a low rating. The first *for* loop

optimizes an objective function *obj* (line 7) for every image in the image test set (*ITS*). The objective function is used to evaluate the fitness of possible feature weight vectors. It is listed in formula 6, with v being the feature weight vector to be tested, n being the number of result images returned by the CBIRs and $score_{\alpha,q}(i'_k)$ being the returned score value for result image i'_k respective to input image q (see formula 3). The function $score'$ is listed separately in formula 5. It is very similar to formula 3 and is used to calculate an known score value for result image i'_k, input image q and feature weight vector v. Here, ${F_\omega}'$ is the set of currently remaining features inside *PossibleFeatures*, f_j is the j-th remaining feature and $\delta_{\omega\mathrm{j}}$ is one of the distance functions used by our internal representation ω. Additionally, the function *scale* normalizes $score'$'s values to the interval $[0, 1]$.

$$score'_{\omega,\mathrm{q},\mathrm{v}}(i') = scale_q[\sum_{j=1}^{|{F_\omega}'|} v_\mathrm{j}\delta_{\omega\mathrm{j}}(f_\mathrm{j}(i'), f_\mathrm{j}(q))] \tag{5}$$

$$obj_q(v) = \sqrt{\frac{\sum_{k=1}^{n}(score_{\alpha,q}(i'_k) - score'_{\omega,q,v}(i'_k))^2}{n}} \tag{6}$$

As mentioned above, the objective function *obj* calculates a distance value for the scoring obtained by using feature weight vector v and the actual scoring of the CBIRs. Larger distance values mean that a tested vector results in self-computed scores less similar to those returned by the CBIRs, i.e. there are larger differences between *score* and $score'$ for the different result images.

Furthermore, algorithm [16,6,5] to find a feature weight vector which most closely resembles the CBIRs' configuration. Multiple optimization algorithms have been evaluated and the results are presented in section 5. This optimization is performed for every image in *ITS*, so *FeatureWeights* consists of vectors with each vector containing the optimal weight values for one image in *ITS* (line 7).

The reason for calculating an optimal feature weight vector for multiple images is to be able to conduct a statistical analysis of these values afterwards, which is done in the second *for loop* starting in line 9. *FeatureWeights* contains multiple weight values for a single feature, one for every image in *ITS* (for example see figure 2 where for f_A three different score settings have been detected). *rateFeature* assigns a rating between 0 and 1 to every feature, depending on different configurable criteria (line 10). Currently, a higher standard deviation of the weight values of one feature for different images results in a reduction in its rating, as does too small an average weight. The ratings of all features are also lowered if the distance values returned by the objective function are higher, which signifies that a weight vector cannot reproduce the CBIR system's behavior well enough. If any feature has a rating smaller than *minRating* (line 11) it is not used again for future executions of the outer *while* loop since a smaller rating suggests a lower probability of a feature being used by the analyzed CBIRs. The outer while is only repeated if at least one feature has been discarded in the current run (line 13). When the *while* loop finishes, depending on whether there are still remaining features in *PossibleFeatures*, an average of the previously computed

Algorithm 1. analyze(ITS, F_ω, CBIRsResults[])

```
1: doContinue ← true
2: PossibleFeatures ← F_ω
3: while doContinue do
4:    doContinue ← false
5:    for i = 1 to i = |ITS| do
6:       q ← ITS[i]
7:       FeatureWeights[i] ← optimize(obj_q)
8:    end for
9:    for i = 1 to i = |PossibleFeatures| do
10:      Ratings[i] ← rateFeature(i, FeatureWeights)
11:      if (Ratings[i] < minRating) then
12:         PossibleFeatures.remove(i)
13:         doContinue ← true
14:      end if
15:   end for
16: end while
17: if PossibleFeatures.isEmpty() then
18:   return null
19: else
20:   return average(FeatureWeights)
21: end if
```

optimal feature weights for the remaining features is returned. Otherwise it is reported that the analyzed CBIRs uses unknown features (lines 17-20).

5 Evaluation

The evaluation section is divided into three parts. The first part addresses the runtime of the implementation of our approach. The second part evaluates the performance of the presented approach respective to the detection of the features that were relevant for the analyzed CBIRs. Finally, the last part focuses on the accuracy of the weight allocation for the detected features.

Our tests used the publicly available image set of the ImageCLEF benchmark [10] which can be obtained at `http://www.imageclef.org/2011`. We used the full set of 20,000 images. As a CBIR system we used Lire [8] which is an open source Java CBIR library containing a good set of implemented features[1]. Furthermore, Lire was chosen as it is an extensible library and could therefore be adapted with little effort.

5.1 Runtime

One possible way of approximating the configuration of a CBIRs is to try a number of feature weight vectors one by one (brute force). Equation 7 represents the number of possible feature weighting allocations, where n stands for the number

[1] `http://www.semanticmetadata.net/lire/`

of features multiplied by the number of feature distances (metrics) and k stands for the granularity of individual weight values. A granularity of 1 would only enable the algorithm to identify whether a feature-metric combination was chosen with a weighting of 100% or 0%. By contrast, a granularity of 100 would make it possible to allocate individual weight values in the weight vector in 1% steps.

$$t(n,k) = \binom{n+k-1}{k} \tag{7}$$

Under realistic circumstances, using a brute force approach would lead to a prohibitively large runtime. The application of different optimization algorithms solved this runtime problem. An implementation of Cuckoo Search [16] as well as an implementation of Particle Swarm Optimization [6], the usage of Multi-Directional Search and the Nelder-Mead Method from [5] all needed approximately 15 seconds for the analysis of one system and delivered very good results. However, about two minutes of additional time was required for extracting the feature vectors from images and pre-calculating feature distance values between images, regardless of which algorithm was used.

5.2 Feature Detection

The evaluation involved the analysis of 500 different configurations (combinations of features and weights) of a Lire-CBIRs. For every one of the cases defined in section 4.1 a test set of 100 different configurations was randomly generated, with each of the five test sets adhering to its respective classification case constraint. Also, for every optimization algorithm 50,000 evaluations of the objective function were performed.

Figure 3 shows the average precision and recall percentages regarding the correct identification of features over the 100 different configurations belonging to case 1. As mentioned in section 5.1 all of the optimization algorithms delivered very good results. The brute force approach using a granularity value of 10 was chosen for comparison as it had a runtime similar to the other optimization algorithms. All of the optimization algorithms returned mostly the same features as the selected features in the CBIRs configuration. This means that for nearly every possible configuration the optimal feature weighting calculated by the algorithms was using all the features known to our internal representation. This is the desired behavior, as our internal representation implemented exactly the same features as the Lire-configurations in this case. False negatives - features which are not marked as detected but are in fact used by the CBIRs - did not occur often. These few false negatives, reflected in the marginal deviation from 100%, were mostly caused by configurations where one of the features used a weight percentage of less or 1%.

Figure 4 shows the average precision and recall percentages over the 100 different configurations of case 2. In this case the internal representation implements more features than the CBIRs provides. Here, false positives - features that are marked as detected but are not in fact used by the CBIRS - can occur in addition to the previously described false negatives. Cuckoo Search as well as

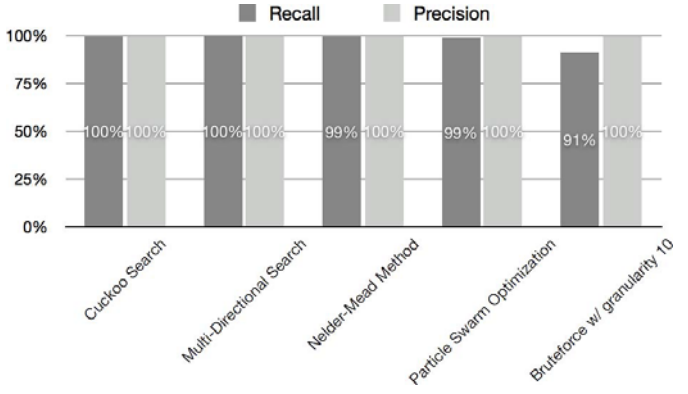

Fig. 3. Precision and Recall: Case 1

Multi-Directional Search also exhibited very good performance. For all the tests every feature used was detected and false positives occurred only in a few cases. The performance of the Nelder-Mead Method and Particle Swarm Optimization were slightly weaker. They both had a small percentage of false negatives, see recall, and a small percentage of false positives, see precision. Those false negatives and positives belonged to tests where only a very small weight value was assigned to a feature. In a real world scenario, though, it would not be very detrimental to the performance of our approach to not be able to correctly identify features with very small weights or to incorrectly assign very small weights to irrelevant features. So these small deviations do not pose a problem.

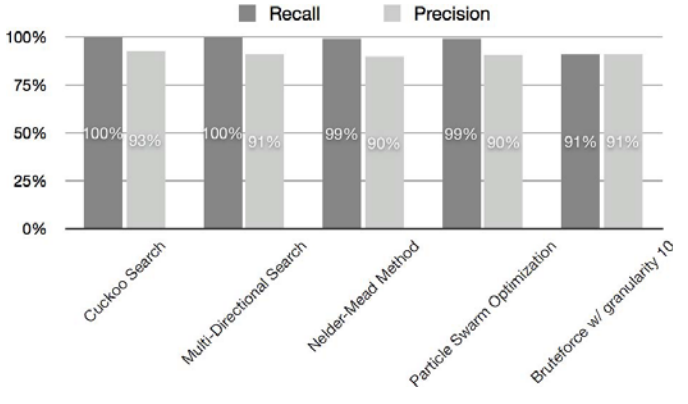

Fig. 4. Precision and Recall: Case 2

Since the focus of the analysis of cases 3 to 5 is not to understand which features were used but rather to discover that the configuration cannot be detected, it would not have been useful to calculate recall and precision values for these cases. That is why in table 1 only the success rates of the different optimization algorithms for the remaining cases are illustrated. A test case was counted as

successful if our implementation returned that it was not able to determine the CBIR system's configuration.

In case 3 our internal representation implemented only a fraction of the features of the CBIRs. In line 2 of table 1 the average success rate over the test runs for every algorithm is shown. Here, each of the algorithms detected (in almost all tests with a probability of 91 %) that the internally implemented features were not a superset of or equal to the CBIR system's features.

In case 4 our internal representation implemented a fraction of the features of the CBIRs as well as additional features which were not implemented by the CBIRs. Very similar to case 3 in most tests, our implemented approach detected that it was not able to identify the CBIR system's configuration due to missing features. As the internal representation had a larger amount of features available to approximate the CBIR system's behavior, in marginal cases a CBIRs's configuration was sometimes incorrectly considered to be detected. The brute force method delivered better results in these cases because it was weaker in terms of optimization precision and was thus more likely to identify a configuration as unknown.

Table 1. Sucess rates of Cases 3, 4 and 5

	Cuckoo Search	Multi-Directional Search	Nelder-Mead Method	Particle Swarm Optimization	Bruteforce (Granularity 10)
Case 3	*91.00%*	*91.00%*	*91.00%*	*91.00%*	*89.00%*
Case 4	88.00%	90.00%	87.00%	87.00%	90.00%
Case 5	100.00%	100.00%	100.00%	100.00%	100.00%

In case 5 the internal representation implemented only features which were not implemented by the CBIRs. As illustrated, all optimization algorithms used as well as the brute force approach were able to detect this classification class.

5.3 Weighting Allocation

In this subsection the weighting allocation performance of our approach is evaluated. This is done by computing the deviation of the detected weighting allocation to the weighting allocation of the CBIRs using the euclidian distance between the weight vectors.

Figure 5 shows a visualization of the deviation for test case 1 for every implemented algorithm. All of the algorithms used did have a small weighting deviation. Cuckoo Search had a deviation of 0.0175, whereas the Multi-Directional Search and the Nelder-Mead Method had a deviation of 0.0217 and 0.0271. PSO and the brute force approach had a deviation of 0.0322 and 0.0637, respectively. In Figure 6 the average deviation for test case 2 is visualized. Again, all of the algorithms do have a small weighting deviation, though mostly slightly larger than in case 1. All of these deviation values are relatively small, meaning that all of the algorithms are able to approximate CBIRs configurations well if all used features are known. Cockoo Search was the best overall algorithm in our tests, though the performance of the various optimization algorithms can be very dependent on their configured parameters.

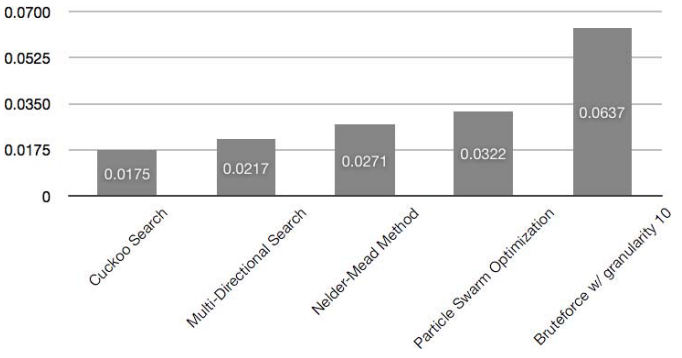

Fig. 5. Averaged weighting deviation in Case 1

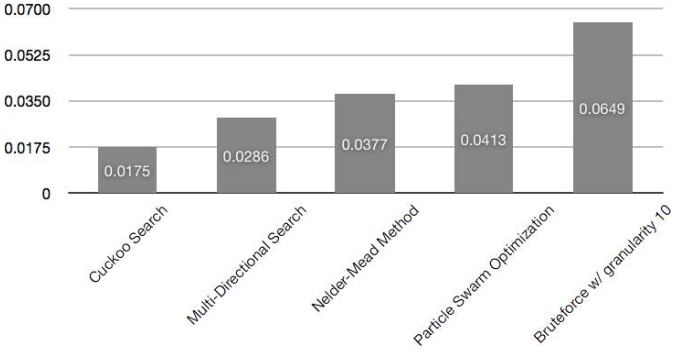

Fig. 6. Averaged weighting deviation in Case 2

All in all, these first test results are already very promising but further improvement and fine tuning of our approach are both necessary.

6 Conclusion

This article presented a novel approach for the detection of the configuration of content based image retrieval (CBIR) systems. The focus of this work was on the correct identification of feature settings and their assigned weights. Related to the combination of a system's configuration and our internal representation, five different classification cases have been highlighted. We demonstrated that our proposed approach is capable of detecting the complete configuration for two cases and is also able to mark the others are as currently not detectable. Moreover, we demonstrated that the problem can be traced back to an optimization problem. The evaluation showed a high accuracy for feature and weight allocation and demonstrated good performance by the use of different optimization algorithms.

Future work will consider the development of feature classes in order to solve the missing classification classes as well. Furthermore, the approach will be

adopted in a distributed search scenario for improving query distribution decisions and result ranking strategies.

References

1. Barton, S., Dohnal, V., Sedmidubsky, J., Zezula, P.: Building self-organized image retrieval network. In: Proceeding of the 2008 ACM Workshop on Large-Scale Distributed Systems for Information Retrieval, pp. 51–58 (2008)
2. Datta, R., Joshi, D., Li, J., Wang, J.: Image retrieval: Ideas, influences, and trends of the new age. ACM Computing Surveys (CSUR) (January 2008)
3. Eidenberger, H.: Evaluation of content-based image descriptors by statistical methods. Multimedia Tools and Applications 35(3), 241–258 (2007)
4. Black, J.A., Fahmy, G., Panchanathan, S.: A Method for Evaluating the Performance of Content-Based Image Retrieval Systems Based on Subjectively Determined Similarity between Images. In: Lew, M., Sebe, N., Eakins, J.P. (eds.) CIVR 2002. LNCS, vol. 2383, pp. 356–366. Springer, Heidelberg (2002)
5. John Ashworth Nelder, R.M.: A simplex method for function minimization. Computer Journal 7, 308–313 (1965)
6. Kennedy, J., Eberhart, R.C.: A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micromachine and Human Science, Nagoya, Japan, pp. 39–43 (1995)
7. Kosch, H., Maier, P.: Content-based image retrieval systems - reviewing and benchmarking. 54 Journal of Digital Information Management (8), 1–21 (March 2010)
8. Lux, M., Chatzichristofis, S.: Lire: lucene image retrieval: an extensible java cbir library. In: Proceeding of the 16th ACM International Conference on Multimedia, pp. 1085–1088. LIRE (2008)
9. Müller, H., Müller, W., Marchand-Maillet, S., Pun, T., Squire, D.M.: Learning feature weights from user behavior in content-based image retrieval. In: Proceedings of the International Workshop on Multimedia Data Mining, Boston, USA, pp. 67–72. ACM (2000)
10. Müller, H., Tsikrika, T.: Global pattern recognition: The imageclef benchmark. IAPR Newsletter 32(1), 3–6 (2010)
11. Smeulders, A., Worring, M., Santini, S., Gupta, A., Jain, R.: Content-based image retrieval at the end of the early years. IEEE Transactions on Pattern Analysis and Machine Intelligence 22(12), 1349–1380 (2000)
12. Stegmaier, F., Döller, M., Kosch, H., Hutter, A., Riegel, T.: AIR: Architecture for Interoperable Retrieval on distributed and heterogeneous Multimedia Repositories. In: Proceedings of the 11th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS 2010), Desenzano del Garda, Italy, pp. 1–4. IEEExplore (2010)
13. Thomas Deselaers, D.K., Ney, H.: Features for image retrieval: an experimental comparison. Information Retrieval 11(2), 77–107 (2007)
14. Veltkamp, R., Tanase, M.: A survey of content-based image retrieval systems. In: Content-based Image and Video Retrieval, pp. 47–101 (2002)
15. Kore, S., Kondekar, V.H., Kolkure, V.S.: Image retrieval techniques based on image features: A state of art approach forcb ir. International Journal of Computer Science and Information Security 7(1), 69–76 (2010)
16. Yang, X., Deb, S.: Engineering optimisation by cuckoo search. International Journal of Mathematical Modelling and Numerical Optimisation 1(4), 330–343 (2010)