Using the Wavelet Transform to Learn from User Feedback

Ilaria Bartolini[1]
ibartolini@deis.unibo.it

Paolo Ciaccia[1]
pciaccia@deis.unibo.it

Florian Waas[1,2]
flw@mx4.org

[1]DEIS – CSITE-CNR, Università di Bologna
40136 Bologna, Italy

[2]CWI, Kruislaan 413
1098 SJ Amsterdam, The Netherlands

Abstract

User feedback has proven very successful to query large multimedia databases. Due to the nature of the data representation and the mismatch between mathematical models and human perception, the query techniques benefit substantially from interactively modifying a query. Typical examples are generalized ellipsoid queries where optimal ratios and orientations of the half-axes are determined by relevance feedback. However, no information about the outcome of a feedback process is stored whatsoever once the process is terminated. Accordingly, the entire feedback loop has to be repeated—starting out with default parameters—if the same query is posed again.

In this paper we present preliminary results on how to preserve feedback results in a space efficient way and *learn* from user feedback. The cornerstone of our system are *multidimensional unbalanced wavelets* that are used to store the parameters determined during the feedback process. Using wavelets lets us not only *store* parameter combinations but also enables us to *predict* parameter settings for queries similar to earlier ones by interpolation: the feedback process for an entirely new query can be started with a parameter setting, usually much closer to the optimal than the default parameters. As a result, after an initial learning phase, feedback is needed for fine tuning only, increasing effectiveness and response time of multimedia databases.

1 Introduction

User feedback in interactive similarity search has been recognized as a highly effective tool. Within a feedback loop, users refine their queries by rating the elements of an answer set according to similarity or dissimilarity. Then, a new answer set is computed using the adjusted query parameters, also often referred to as feature weights. This process is repeated until the user is satisfied with the results. So during a feedback loop the default parameters used by the query engine are gradually adjusted to fit the user's needs (see e.g. [1]).

Different models for the analysis of user feedback, including elliptic, concave, and disjunctive queries have been proposed. Typical representatives include Rocchio's feedback mechanism, MARS, MindReader, or Falcon, to name just a few [2, 1, 3, 4]. These methods converge with an increasing number of iterations to the optimal adjustment of feature weights.

However, feedback mechanisms come with two drawbacks:

1. Depending on the query, it may require numerous iterations before acceptable setting, i.e. answer set, is found.

2. Once the feedback loop of a query is terminated, no information about this particular query is retained for re-use in further processing. Rather, for further queries, the feedback process is started anew with default values. Even in the case that a query object has already been used in an earlier feedback loop, all iterations have to be repeated.

In this paper, we address the question how to store and maintain feedback data efficiently not only in order to re-use data for recurring queries but also to predict feature weights for new queries.

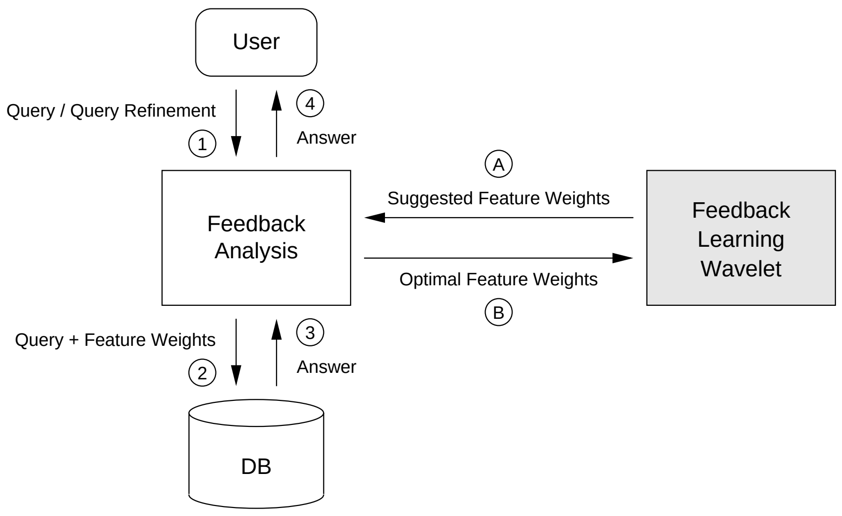

Figure 1: Feedback driven retrieval system with Feedback Learning Wavelet (FLW)

Figure 1 shows the components of a retrieval system with relevance feedback, including the desired component for managing feedback data beyond the lifetime of single queries: First, a query is submitted by the user (1). The *feedback-learning wavelet* (FLW) is consulted and suggests feature weights, which are either previously obtained feedback in case of a recurring query or an approximation based on feedback for related queries (A). The query together with the suggested feature weights is passed to the database (2), which retrieves a corresponding answer set (3,4). The steps 1,2,3, and 4 are repeated according to the feedback analysis until the user is satisfied with the results. Finally, the feature weights are sent to the feedback-learning component to be stored for further queries (B).

2 Model for the Problem

The assumption underlying feedback driven querying is that there exists an optimal combination of the individual feature weights for every query point in the feature space.[1] At the end of a feedback loop we obtain this optimal value for the given query object. In more abstract terms, we have to deal with an *a priori* unknown parameter function which is step by step revealed with more and more feedback data becoming available. We denote this parameter function by $f_{opt} : R^d \to R^n$, for a d-dimensional real-valued feature space, and where n represents the number of parameters of the function. Theoretically, this function can be of arbitrary complexity—but has to respect the abilities of the query engine used, i.e. if the query engine, for example, supports only ellipsoid queries, it is of no use to specify concave answer sets. With this abstraction in mind, we can reformulate the aim of our work as approximating the parameter function given the feedback data gathered so far.

We identify the following requirements for such an approximation to be successful. The approximation should

1. interpolate the available feedback data; the accuracy of the interpolation is an input parameter of the system;
2. efficiently handle updates, i.e. incorporate additional feedback data with only local re-computation of the approximation;
3. be able to evolve over time in that the accuracy of the approximation increases with increasing feedback data. However, feedback data that does not contribute significant changes should be automatically discarded.

Since these desiderata represent somewhat contradictory optimization goals, we need to find a feasible trade-off.

[1]By feature weights we mean not only weighted euclidean metrics but *any* given metric including generalized ellipsoid queries and concave queries as introduced in [4].

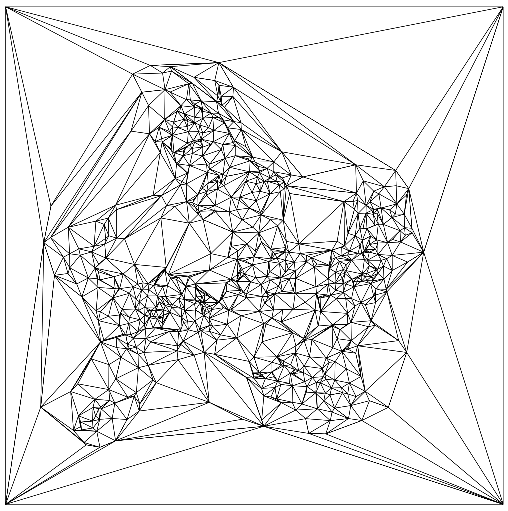

Figure 2: Triangulation of a 2-dimensional data set

3 Maintaining Feedback

As we pointed out above, the original problem can be interpreted as one of approximating a multidimensional, *a priori* unknown parameter function. Though several standard approximation schemes known from numerical mathematics are potential candidates, only few meet our requirements satisfactorily. We chose Multi-resolution Analysis, a particular form of the Wavelet Transform, as it fits our needs best (see e.g. [5]).

Before detailing the single steps, we simplify the problem by decomposing the parameter function $f_{opt} : R^d \to R^n$ into n functions of the type $f_i : R^d \to R$, which can be dealt with independently.

3.1 Feedback Learning Wavelet

The Lifting Scheme is a recent development in the area of approximation theory, which is distinguished by both its simplicity as well as its effectiveness [6]. Due to space limitations we outline only the main principle: Given a set of data points, groups of $2k$ neighbors are used to interpolate using polynomials of degree $2k - 1$, the simplest of which is linear interpolation $(k = 1)$. Repeated removal of data points and local re-computation of the interpolation eventually provides the well-known multi-resolution scheme. In the case of linear interpolation we obtain a Haar wavelet. To use the Lifting for incremental interpolation, the principle is simply inverted and instead of removing points, additional points are added. For a detailed description of the Lifting scheme and the associated interpolation techniques, we refer the interested reader to [7].

Organizing the feature space. In contrast to typical application areas of the Wavelet Transform such as image compression, feedback data is not equidistantly distributed but depends on the queries posed and the objects available in the database. As a result, we need to organize the feature space in unbalanced intervals. To do so, we use simplices of dimension d, i.e. triangles in the 2-dimensional space, tetrahedrons in 3-dimensions, and so on.

To better illustrate this point, consider a 2-dimensional feature space; all data objects correspond to points in the plane. No matter the number of objects and their location we can superimpose a triangular mesh so that each data point is a corner of at least one triangle. Figure 2 shows a triangulation for a randomly generated data set of 2000 points.

Since the simplices are disjoint, their number is on average much lower than in the worst possible case. For example, in the 2-dimensional case of a triangulation, as a fundamental result of graph theory, each point is connected to 6 other points on average. As 3 points make up a simplex, the number of triangles is on average twice the number of data points (see also next section for an experimental assessment of this point).

Predicting parameters. Each wavelet stores only the exact function values for data points that have been inserted so far. No information on how to approximate function values for other points is contained. We use a linear interpolation schema to this purpose. Thus, predicting parameter values for a query point means to find the enclosing simplex and compute the interpolation from the $d+1$ corners of the simplex. Consider $d = 2$ and let (x_i, y_i) be the corners of the simplex (i.e. triangle) enclosing the query point (x, y). Moreover, let $z_i = f(x_i, y_i)$. Solving the equation

$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = 0$$

provides the value $z = f(x, y)$. Higher dimensions are dealt with in an analogous way. Additionally, locating the enclosing simplex can be achieved efficiently using a spatial index structure.

Updates. When feedback data is provided, we first need to look up the enclosing simplex like above. If the feedback data diverges from the predicted value z (see above) by more than a given threshold δ, this very simplex is subdivided into $d+1$ simplices around the new data point by connecting each of the $d+1$ corners with the new data point. Note, both, subdivision and insertion are in $O(1)$, which is of particular importance if a large amount of feedback data is to be maintained.

3.2 Example

Figure 3 demonstrates the evolution of the approximation of a 2-dimensional parameter function. The individual plots show the approximation after 50, 250, 500, and 1000 incremental updates. The figures underline the two

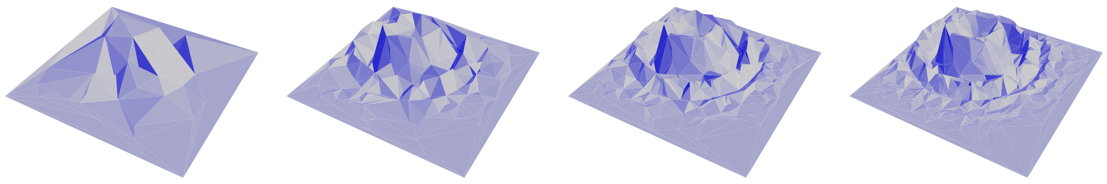
Figure 3: Approximation of a parameter function after incorporating 50, 250, 500, and 1000 feedback points

major advantages of our method: firstly, the approximation converges very quickly towards the actual parameter function. Secondly, the approximation is self-maintaining in the sense that only data points whose contribution is significant are included into the function—i.e. in areas where the parameter function has low frequency (foreground) only few points are added. Oversampling in the form of storing data points which do not contribute significantly to the shape of the function is automatically prevented (see also next section).

4 Preliminary Results

The results we present here give an impression of the potential of the method proposed. The assessment is not comprehensive but covers only the major aspects as large parts of the project are still under development.

As measure of performance improvement achieved by adding the FLW to a feedback-driven retrieval system, we concentrate on the precision of the first answer set for each query. To that end, we simulate the feedback process by using a synthetic parameter function which provides feedback for any given query point.

We run k-nearest-neighbor searches on a 2-dimensional data set of 10000 points using different configurations. The data is taken from 15 Gaussians—Figure 2, used earlier, actually shows a uniform sample from this data set. We constructed a synthetic parameter function $f : R^2 \to R$ by combining periodical and kernel functions, which represents the optimal query parameters in our experiment (see Fig. 3). We use this parameter as ratio of half-axes of the query ellipsoid in the actual query processing, thus implementing a weighted euclidean metric. Knowing the optimal parameter for a given query, we can compute the k nearest neighbors for the actual weighted euclidean metric and compare them with the counterpart, the k nearest neighbors found with the parameter *predicted* by the FLW.

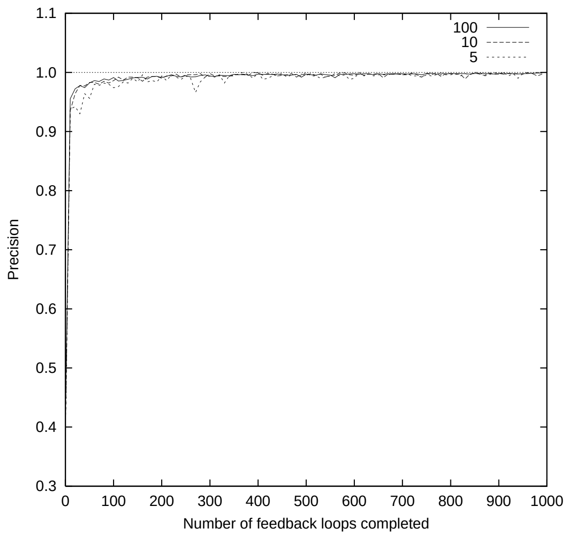

Figure 4: Precision of initial answer set

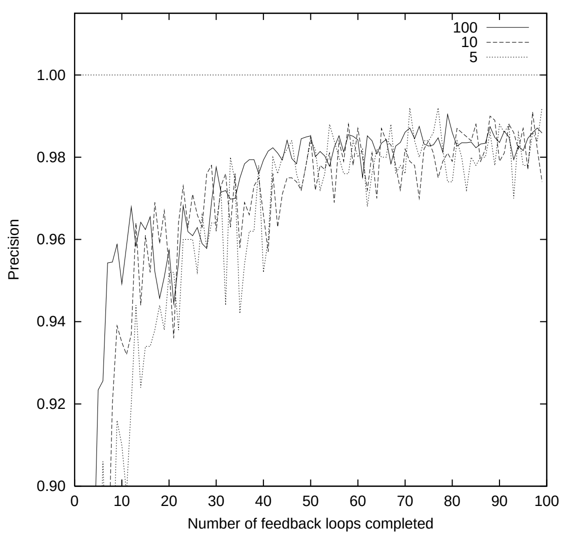

Figure 5: Precision (close-up)

Precision. The precision is computed relative to the optimal k nearest neighbors:

$$precision = \frac{1}{k} \cdot \textit{number of elements common to both sets}$$

Since we always compute k nearest neighbors for the optimal answer, the recall is not relevant at this point.

In the following experiments we varied the number of nearest neighbors k and the threshold δ, which determines whether new feedback data is inserted in the FLW or not. In Figure 4, the precision is plotted as function of the number of feedback loops computed, i.e. when synthetic feedback is provided. k was chosen as 5,10, and 100, respectively. As the plot illustrates, the precision quickly shoots up from below 60% with default parameters (first data point) to above 95%. After as little as 20 feedback loops completed, the prediction by the FLW achieves a precision of more than 90%. In Figure 5, a close-up of the first 100 queries is depicted. The plot also shows that the precision is only little impacted by the number of nearest neighbors we compute: the three curves are almost coincident.

In further experiments, we found that δ has only marginal influence on the precision as long as it is kept below 10–20% of the entire range of the function. In details, let $\delta = a + \alpha \cdot (b - a)$, where a and b represent, respectively, the minimal and maximal values of the function f, and α is a percentage. Thus, if $a = 0.2$ and $b = 10$, setting $\alpha = 20\%$ would lead to $\delta = 2.16$. Given δ, a new feedback point is inserted in the FLW only if its difference from the predicted value is larger than $\delta/2$.

Resource Efficiency. When it comes to resource usage, δ plays a far more important role. Figure 6 shows the number of points stored in the FLW as function of the number of queries, with $\alpha = 0.2\%, 2\%, 5\%$, and 10%, respectively.

As the graph details, at the beginning almost every data point is inserted but with increasing number of queries the number of additional values to be incorporated decreases quickly. The lower the threshold, the more points need to be maintained. For larger thresholds, this number converges rapidly not exceeding a few hundreds even after 10000 updates attempts.

An aspect, very important for the scaling to higher dimensions, is the ratio of simplices to data points. In Figure 7 this ratio is shown to converge quickly to the expected value, 2 in this case. Here, lower thresholds cause quicker convergence as approximations with lower thresholds induce a more regular decomposition into simplices. Conversely, the number of simplices is in $O(\textit{number of points} \cdot d)$, d being the dimensionality of the feature space.

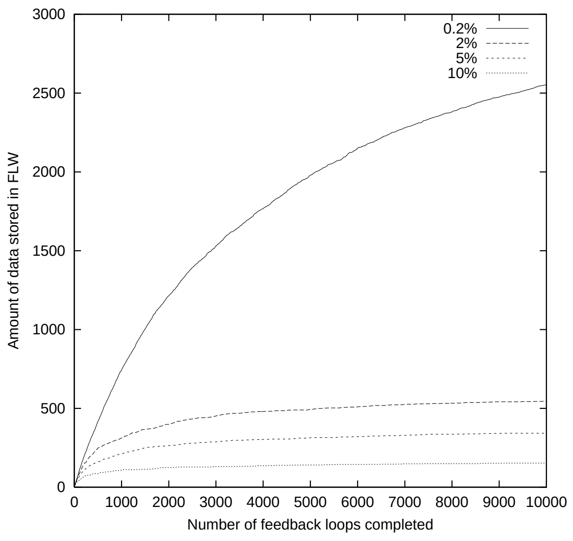

Figure 6: Points stored in FLW

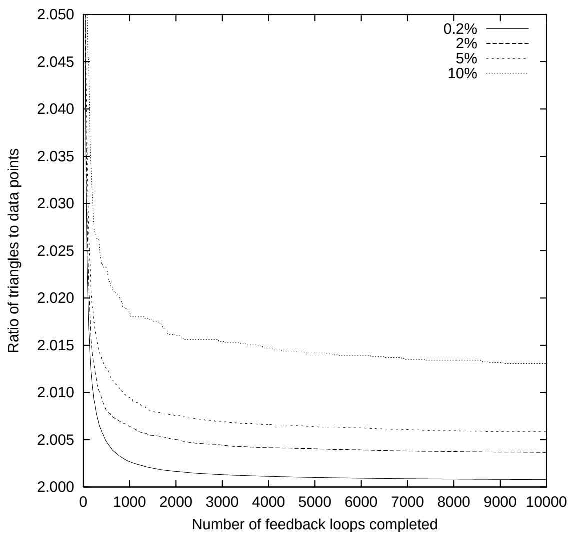

Figure 7: Ratio of simplices to data points

5 Conclusion

We presented Feedback Learning Wavelets (FLW), a technique to store and maintain feedback results in a resource efficient way. The re-use of feedback results for recurring queries and the approximation in case of new queries can help speed up the interactive retrieval process greatly. As our initial results show, after only a few tens of queries, the precision of the first answer to a query exceeds already 95% before the users is consulted for feedback. Using our technique, large parts of the feedback process can be bypassed and high quality results are obtained much earlier.

The major issue our future research is concerned with is scaling to high dimensional spaces. Each of the techniques presented are linear in the number of points and/or the dimensionality, with computing the determinant for the approximate parameter values being the only exception. However, practical methods are available for computing determinants of matrices up to sizes that exceed the dimensionality of typical feature spaces by far. Another issue we intend to investigate is as to how the FLW can be used in non-euclidean metric spaces.

References

[1] M. Ortega, Y. Rui, K. Chakrabarti, S. Mehrotra, and T. Huang. Supporting similarity queries in MARS. In *Proc. of the Int.'l Conference on Multimedia*, pages 403–413, Seattle, WA, USA, November 1997.

[2] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, 1988.

[3] Y. Ishikawa, R. Subramanya, and C. Faloutsos. MindReader: Querying Databases Through Multiple Examples. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 218–227, New York, NY, USA, August 1998.

[4] L. Wu, C. Faloutsos, K. Sycara, and T. Payne. FALCON: Feedback Adaptive Loop for Content-Based Retrieval. In *Proc. of the Int'l. Conf. on Very Large Data Bases*, pages 297–306, Cairo, Egypt, September 2000.

[5] G. Kaiser. *A Friendly Guide to Wavelets*. Birkhäuser, Boston, Basel, Berlin, 1994.

[6] W. Sweldens. The Lifting Scheme: A Custom-design Construction of Biorthogonal wavelets. *Appl. Comput. Harmon. Anal.*, 3(2):186–200, 1996.

[7] W. Sweldens and P. Schröder. Building Your own Wavelets at Home. In *Wavelets in Computer Graphics*, pages 15–87. ACM SIGGRAPH, 1996.